VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Autokonfigurační směrovací systém pro SW PBX Asterisk

System for Routing Autoconfiguration in SW PBX Asterisk

2013 **David Rykala**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání bakalářské práce

Student: **David Rykala**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R059 Mobilní technologie

Téma: Autokonfigurační směrovací systém pro SW PBX Asterisk
System for Routing Autoconfiguration in SW PBX Asterisk

Zásady pro vypracování:

Softwarová PBX Asterisk je v dnešní době jedním z nejčastěji implementovaných řešení pro komunikaci prostřednictvím VoIP, ISDN a případně i analogové telefonie. Penetrace trhu tímto řešením v posledních 5 letech masivně vzrostla a to zejména díky zájmu jednotlivých subjektů na konvergenci hlasových a datových komunikací z důvodu úspory nákladů. Při instalaci každé ústředny je třeba konfigurovat komunikační kanály a uzly, přes které bude procházet následná komunikace. Tento krok lze mnohde považovat za rutinní a je proto výhodné mít k dispozici nástroj, který řadu opakujících se kroků automatizuje. Studie, algoritmický návrh a rozbor případných implementačních směrů tohoto nástroje jsou cílem této bakalářské práce, která by spolu s porovnáním navrhovaného řešení a technologií jako ENUM a DUNDI měla poskytnout ucelený pohled na problematiku směrování hovorů do destinací, které nejsou administrátorem explicitně zadávány.

Body zadání:

1. Architektura a filosofie softwarové PBX Asterisk (hardware, kanály, dialplán).
2. Směrování hovorů v rámci Asterisk dialplánu.
3. Meziasterisková výměna informací pro automatické vytváření trunkových spojů.
4. Integrace vyměňovaných informací do aktivního dialplánu.
5. Srovnání navrhovaného řešení s technologiemi ENUM a DUNDI.

Seznam doporučené odborné literatury:

[1] Meggelen, J.V., Smith, J., Madsen, L.: Asterisk: The Future of Telephony. O'Reily Media, 2005, ISBN: 0-596-00962-3.
[2] Madsen, L., Meggelen, J.V., Bryant, R.: Asterisk: The Definite Guide. O'Reily Media, 2011, ISBN: 978-0-596-51734-2.
[3] Wintermeyer, S. Bosch, S.: Practical Asterisk 1.4 and 1.6: From Beginner to Expert. Addison-Wesley Professional, 2009, ISBN: 978-0-321-52566-6.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Rozhon**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

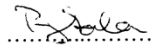
prof. RNDr. Vladimír Vašinek, CSc.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Dne: *3.5.2013*



podpis studenta

Poděkování

Rád bych poděkoval Ing. Janu Rozhonovi za odbornou pomoc a konzultaci při vytváření této bakalářské práce.

Abstrakt

Cílem této bakalářské práce je studie, algoritmický návrh a rozbor případných implementačních směrů autokonfiguračního systému pro softwarovou PBX Asterisk. Při instalaci každé ústředny je třeba konfigurovat komunikační kanály a uzly, přes které bude procházet následná komunikace. Tento krok lze mnohde považovat za rutinní a je proto výhodné mít k dispozici nástroj, který řadu opakujících se kroků automatizuje. Bakalářská práce obsahuje i popis simulace navrhovaného řešení spolu se všemi zdrojovými kódy, které se nachází v příloze. Je zde popsán aktivní dialplán pro integraci vyměňovaných informací. Závěr práce se věnuje srovnání navrhovaného řešení s technologiemi ENUM a DUNDI.

Klíčová slova

Asterisk, dialplán, DUNDi, ENUM, meziasterisková výměna informací, PBX, SIP, směrování hovorů, VoIP

Abstract

The aim of this thesis is the study, the algorithmic design and the analysis of possible directions of implementation of Auto-configuration for Asterisk software branch. The communication channels and nodes must be configured to install each panel. The communication will pass through these channels and nodes. This step can often be considered routine and is therefore beneficial to have an available tool that automates many repetitive steps. Bachelor thesis contains a description of the simulation of the proposed solutions together with all source code, which is located in the annex. There is also described active dial-plan for the integration of exchanged information. The conclusion is devoted to a comparison of the proposed solution with ENUM technology and DUNDi.

Key words

Asterisk, dial-plan, DUNDi, ENUM, asterisk´s exchange information, PBX, SIP, call rating, VoIP

Seznam použitých zkratek

Zkratka	Anglický význam	Český význam
ACD	Automatic Call Distribution	Efektivní správa příchozích hovorů
ADPCM	Adaptive Differential Pulse Coded Modulation	Adaptivní diferenční pulzní kódová modulace
ALSA	Advanced Linux Sound Architecture	Linuxová komponenta obsluhující zvuková zařízení
API	Application Programming Interface	Rozhraní pro programování aplikací
BASH	Bourne Again Shell	Příkazový procesor UNIXu
BRI	Basic Rate Interface	Je ISDN konfigurace určená pro použití účastnických linek
CELP	Code-excited linear prediction	Algoritmus pro kódování řeči
CPU	Central Processing Unit	Procesor
CS-ACELP	Conjugate Structure Algebraic Code-Excited Linear Prediction	Algoritmus pro kódování řeči
DAHDI	Digium/Asterisk Hardware Device Interface	Rozhraní k telefonním zařízením
DNS	Domain Name System	Hierarchický systém doménových jmen
DPCM	Differential pulse-code modulation	Diferenciální pulzní kódovaná modulace
DUNDi	Distributed Universal Number Discovery	Distribuovaný směrovací systém
ENUM	TElephone NUmber Mapping	Soubor protokolů pro sjednocení telefonního systému s internetem
FXO	Foreign eXchange Subscribe	Port používaný analogovými telefonickými linkami
FXS	Foreign eXchange Office	Port používaný analogovými telefonickými linkami
GSM	Global System for Mobile communications	Globální systém pro mobilní komunikace
HTTP	Hyper Text Transfer Protocol	Internetový protokol určený pro výměnu hypertextových dokumentů

HW	Hardware	Fyzické existující technické vybavení počítače
I/O	Input/Output	Vstup/Výstup
IAX	Inter-Asterisk Exchange Protocol	Komunikační protokol Asterisku
IP	Internet Protocol	Internetový protokol
ISDN	Integrated Services Digital Network	Digitální síť integrovaných služeb
ITU	International Telecommunications Union	Mezinárodní telekomunikační unie
IVR	Interactive Voice Response	Interaktivní hlasová odezva
JSON	JavaScript Object Notation	Způsob zápisu dat (datový formát) nezávislý na počítačové platformě
LCR	Least Cost Routing	Systém pro směrování cestou nejnižších nákladů
LD-CELP	Low Delay Code Excited Linear Prediction	Algoritmus pro kódování řeči
LPC	Local Procedure Call	Volání lokální procedury
MAC	Media Access Control	Jedinečný identifikátor pro síťová zařízení
MCU	Multipoint Control Unit	Jednotka pro řízení konferenční spojení
MGCP	Media Gateway Control Protocol	Signalizační protokol řídící hlasové brány
MOS	Mean Opinion Score	Subjektivní metoda hodnocení kvality hovoru a kodeku
NAT	Network Address Translate	Překlad adres
OSS	Open sound system	Dostupné zvukové rozhraní
PBX	Private Branch eXchange	Pobočková ústředna
PCI	Peripheral Component Interconnect	Počítačová sběrnice pro připojení periferií k základní desce
PCM	Pulse-code modulation	Pulzní kódová modulace
POTS	Plain Old Telephone Service	Port používaný analogovými telefonickými linkami
PRI	Primary Rate Interface	Standardizovaná telekomunikační úroveň služeb v ISDN

PSTN	Public Switched Telephone Network	Veřejná komutovaná telefonní síť
QoS	Quality of Service	Kvalita služby
RAS	Registration, Admission, Status	Signalizační protokol
RTCP	Real Time Control Protocol	Řídící protokol
RTP	Real Time Protocol	Komunikační protokol
SCCP	Skinny Call Control Protocol	Proprietální signalizační protokol
SDP	Session Description Protocol	Internetový protokol určený k popisu vlastností relace
SF	Software	Sada všech počítačových programů používaných v počítači
SIP	Session Initiation Protocol	Signalizační protokol
SMTP	Simple Mail Transfer Protocol	Internetový protokol pro přenos zpráv elektronické pošty
TCP	Transmission Control Protocol	Spojově orientovaný řídící protokol transportní vrstvy
TDM	Time-division multiplexing	Multiplex s časovým dělením
TLS	Transport Layer Security	Protokol pro zabezpečenou komunikaci na Internetu
TTL	Time to live	Číslo, které omezuje dobu platnosti dat nebo počet průchodů paketů
UA	User Agent	Uživatelský agent
UDP	User Datagram Protocol	Uživatelský datagramový protokol transportní vrstvy
UNISTIM	Unified Networks IP Stimulus	Telekomunikační protokol
VoFR	Voice over Frame Relay	Přenosu hlasu přes síť Frame Relay
VoIP	Voice over Internet Protocol	Internetová telefonie
VPN	Virtual private network	Virtuální privátní síť

Obsah

Seznam tabulek:

Seznam obrázků:

1 Úvod

Softwarová PBX Asterisk je v dnešní době jedním z nejčastěji implementovaných řešení pro komunikaci prostřednictvím VoIP, ISDN a případně i analogové telefonie. Jedná se o open-source softwarovou PBX, běžící na operačních systémech Unix a Linux. Při instalaci každé ústředny je nutno nakonfigurovat uzly a komunikační kanály, přes které se bude komunikovat. Tato práce se právě zabývá návrhem autokonfiguračního směrovacího systému, který řadu opakujících kroků automatizuje. V teoretické části se rozepisuji o technologii VoIP a nejpoužívanějších protokolech, které tato technologie používá. Dále se zde nachází seznámení se softwarovou PBX Asterisk. Následuje již zmíněný autokonfigurační směrovací systém, který je hlavním cílem této práce. Obsahuje popis rozvržení souboru sip.conf a experimentální topologie pro využití tohoto systému. Návrh tohoto systému byl velmi komplexním problémem, a proto byl rozdělen do třech menších celků. Konkrétně se jedná o server, klienta a hlídače. Jednotlivé celky jsou stručně popsány a zobrazeny pomocí vývojového diagramu. Následuje postup simulace navrhovaného řešení. Nechybí zde popis a ukázka navrženého dialplánu. V poslední řadě funkční ukázka průběhu hovoru v Asterisku pomocí informací vyměněných autokonfiguračním směrovacím systémem. Závěr práce je věnován srovnání navrhovaného řešení s technologiemi ENUM a DUNDI, kde jsou určeny výhody a nevýhody tohoto řešení.

2 Voice over IP

VoIP neboli Voice over Internet Protocol je soubor technologií, které umožňují přenos hlasu přes protokol IP. Jde o alternativu pro klasickou telefonii, založenou na použití sítí s přepojováním okruhů přes veřejnou telefonní síť. Mezi výhody této alternativy patří, že se datové sítě šíří a rozvíjejí daleko rychleji než telefonní sítě. VoIP může uskutečňovat hlasový provoz po vlastních datových sítích, čehož nejvíce využívají rozsáhlé a geograficky rozdělené společnosti. VoIP se neustále vyvíjí a snaží se o sjednocení komunikačních standardů a vytvoření sítí s integrovanými službami, které budou moci přenášet data, hlas nebo video po jediné infrastruktuře. Hlavní výhodou je nízká cena, díky redukovaným nákladům na komunikační infrastrukturu a její správu. Když jsme si uvedli výhody, musíme se zmínit i o nevýhodách jako jsou např. snížení spolehlivosti a dostupnosti služby, ozvěna, bezpečnostní rizika jako zneužití VoIP systému nebo obtížnost implementace ostatních VoIP služeb. [1] [2]

VoIP lze využít téměř všude, kde existuje připojení k datové síti resp. Internetu. Kromě připojení k Internetu je potřeba zařízení, pomocí kterého se bude telefonovat. Může být použit vlastní IP telefon nebo ponechán stávající telefon z pevné linky, pokud je doplněn VoIP bránou. Pro telefonování lze také využít tzv. VoIP softwarových aplikací jako jsou SJphone, X-Lite, Skype a mnoho dalších, které stačí nainstalovat na počítač, notebook, tablet či chytrý telefon. Podmínkou je zde opět připojení k Internetu a nutnost zapnutí přístroje i telefonu. Toto telefonování se již více méně neliší od klasické telefonie, jak jsem zmínil výše. Ve VoIP funguje přenositelnost čísel a zákazník si tak může ponechat své telefonní číslo, které využíval u klasické telefonie. Může také pohodlně volat do pevných a mobilních sítí, kde ovšem již tyto hovory nemohou být zdarma, ovšem ceny hovoru jsou stále nižší než při voláni z pevné linky. Tato technologie má velké využití např. v různých firmách díky bezplatnému telefonickému propojení s firemními pobočkami, vazbě na informační systémy nebo nahrávaní hovorů. [3] [4] [11]

Komunikační síť musí poskytovat různé druhy služeb v požadované kvalitě. Zde se zavádí pojem kvalita služeb neboli QoS (Quality of Service). Klade se velký důraz na doručitelnost dat bez jejich ztráty s minimálním zpožděním. V současnosti je toho dosaženo v IP sítích díky třem faktorům, jako pokroky v kompresních technikách snižující nároky na přenosové pásmo, navyšování přenosové kapacity a zavedením nástrojů QoS v IP síti. [1]

Ve VoIP se data přenášejí pomocí různých protokolů. Jsou rozděleny na komunikační a signalizační protokoly. Mezi komunikační protokol se řadí RTP protokol, který je doplněn řídícím protokolem RTCP. RTP protokol podporuje přenos dat v reálném čase a využívá se pro přenos audio a video provozu. Signalizačním protokolů vévodí protokol SIP. Druhý nejznámější je protokol H.323, který je nejstarší široce používaný a díky vzniku SIP se v dnešní době příliš nepoužívá. IAX (Inter-Asterisk eXchange) patří mezi otevřené protokoly, čili jej může kdokoliv stáhnout a využít ve svém projektu. Byl vyvinut pro účel komunikace s Asterisk servery, ale není limitován pouze na Asterisk. Mezi méně známé protokoly patří MGCP, Skinny/SCCP a UNISTIM. MGCP neboli Media Gateway Control protokol používá centralizovaný model. Skinny Client Control Protokol je proprietální Cisco protokol pro Cisco VoIP zařízení a je standardním protokolem pro koncová zařízení v Cisco Call Manager PBX. Protokol UNISTIM je zase proprietální protokol od firmy Nortel. [2] [5] [6]

2.1 H.323 protokol

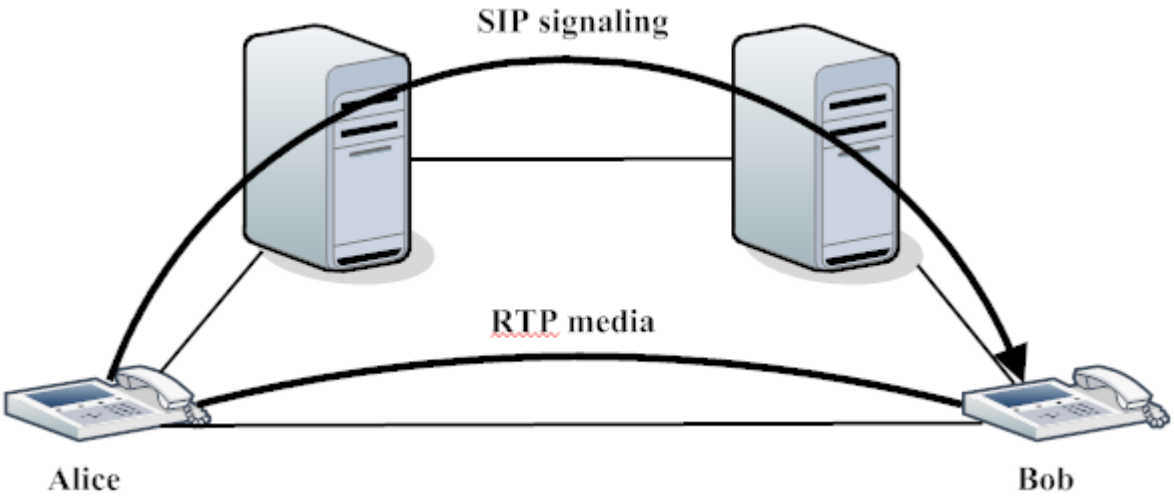
H.323 je nejstarším široce používaným protokolem a byl vyvinut organizací ITU (International Telecommunication Union) v květnu 1996 za účelem přenosu hlasu, videa, datové a faxové komunikace přes IP. Časem se vyvíjí mnoho dalších verzí a doplňků, které přidávají nové funkce. H.323 využívá ke komunikaci protokol RTP, který neobsahuje podporu šifrování. Přenos lze zašifrovat pomocí vytváření VPN tunelů mezi koncovými body spojení. Nevýhodou však je složitější organizace tunelů a vyšší nároky na hardware. Velkým problémem je komunikace skrz NAT, kde dochází k překladu adres. Pokud máme několik klientů za NAT zařízením, potřebujeme gatekeepera pracujícího v proxy módu, který slouží jako brána a pomáhá pro překlad adres v NAT. H.323 protokol, ale začíná pomalu upadat a na výsluní se dostává novější protokol SIP. [6]

H.323 se skládá ze 4 základních komponentů a to Terminál, Gateway (brána), Gatekeeper (vrátný) a MCU (jednotka pro řízení konferenčního spojení). Terminál je základní a taky jediná povinná komponenta. V reálném čase využívá obousměrnou komunikaci. Musí podporovat audio služby a umí komunikovat do jiných sítí s různými terminály. Gateway zabezpečuje spojení H.323 sítě a slouží jako překladač protokolů mezi různými sítěmi. Gatekeeper si lze představit jako mozek celé sítě. Stará se o důležité služby jako autorizace, autentizace, překlad čísel na IP, účtování služeb, směrování hovorů apod. Nejčastěji se GK realizuje softwarem. MCU zajišťuje komunikaci tří a více terminálů a převádí decentralizované spojení na centralizované a naopak. H.323 síť se skládá z těchto komponentů, kdy může obsahovat libovolné množství terminálů, MCU a Gateway, ale vždy pouze jednoho Gatekeepera na jednu zónu. Jedná se pak o H.323 zónu a je třeba zdůraznit, že toto dělení je pouze logické. Velmi často mohou být komponenty integrovány i do jednoho produktu. [1]

Protokol H.323 v sobě zahrnuje kódování zvuku, kódování obrazu, H.225 signalizaci volání a H.225 RAS což slouží pro sestavení spojení a zabezpečení komunikace, řídící signalizaci H.245 pro řízení toku dat, RTP a RTCP protokoly pro samotný přenos hlasu a videa v reálném čase. [1]

2.2 SIP protokol

SIP (Session Initiation Protocol) je signalizační protokol pro sestavení, dohled a ukončení spojení mezi účastníky. Svojí strukturou je podobný protokolům HTTP nebo SMTP. Mezi jeho výhody patří široká podpora a flexibilita, ale především jednoduchost, díky čemuž dokázal sesadit protokol H.323 z trůnu a stal se nejvíce používaným protokolem. Jedná se o protokol aplikační vrstvy, který využívá ke komunikaci port 5060. Je přenášen zejména přes UDP protokol, ale lze využít i TCP protokol transportní vrstvy. SIP využívá RTP (Real Time Protocol) k přenosu médií mezi jednotlivými prvky. K vysvětlení pomůže následující obrázek 1 a popisek. [1] [7]



Obrázek 1: *Vysvětlení použití 2.1 [7]*

„Pokud Alice chce volat Boba, telefon Alice kontaktuje její proxy server a ten se pokusí najít Boba (přes jeho proxy server). Jakmile dojde k vybudování spojení, probíhá komunikace již přímo mezi telefony bez zatěžování prostředků serverů.“ [7]

SIP se skládá z následujících komponent. User Agent a Servery. User agents jsou koncová zařízení SIP sítě a starají se o navázání spojení s ostatními User agents. Úkolem serveru je zprostředkovat kontakt mezi volajícími a volanými, což ale nevylučuje přímý kontakt bez účasti serverů. Rozlišují se tři typy serverů: Proxy server, Redirect server a Registrar server. [1]

Proxy server – přijímá žádosti o spojení od UA nebo proxy serverů a předává je dalšímu serveru nebo volanému UA pokud je v rámci spravované domény. [1]

Redirect server – stejně jako proxy server přijímá žádosti o spojení od UA nebo proxy serverů, ale nepředává je dále volanému, nýbrž předává informaci, komu žádost poslat, tak aby se dostala k volanému. [1]

Registrar server – tento server přijímá registrační žádosti od UA a poté podle nich aktualizuje databázi koncových zařízení spravované v rámci domény. [1]

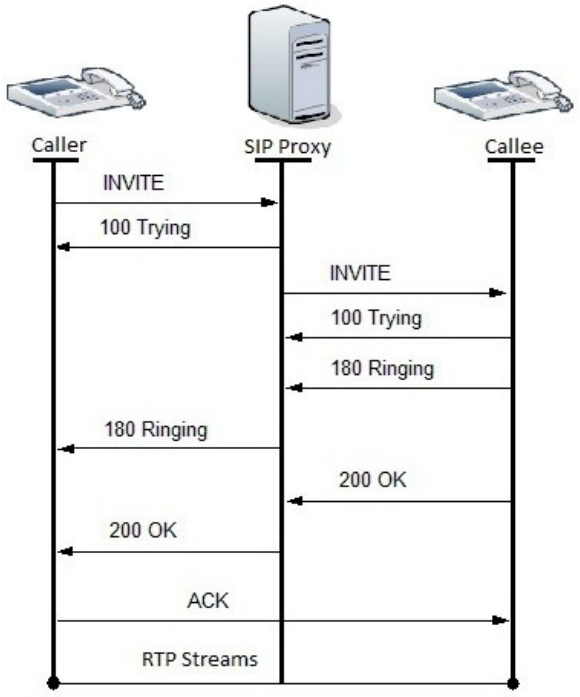
Komunikace mezi SIP komponenty se provádí pomocí žádostí. Mezi šest základních žádostí patří:

- INVITE – žádost o sestavení spojení
- ACK – potvrzení zahájení spojení
- BYE – ukončení spojení při hovoru
- CANCEL – ukončení spojení před jeho navázáním
- REGISTER – registrace User agenta
- OPTIONS – dotaz na možnosti serveru, aniž by se sestavilo spojení

Odpovědi na SIP žádosti jsou zprávy s číselným kódem podobné HTTP odpovědím. Člení se také do šesti skupin:

- 1XX – informační zprávy („100 Trying“, „180 Ringing“)
- 2XX – úspěšné ukončení žádosti („200 OK“)
- 3XX – přesměrování („302 Med Temporarily“, „305 Use Proxy“)
- 4XX – chyba na straně klienta („403 Forbidden“)
- 5XX – chyba na straně serveru („500 Server Internal Error“, „501 Not Implemented“)
- 6XX – globální selhání („606 Not Acceptable“)

Na následujícím obrázku 2 je ukázka spojení prostřednictvím Proxy serveru.



Obrázek 2: *Tok SIP zpráv v rámci SIP Proxy 2.2 [2]*

SIP také implementuje šifrovaný přenos pomocí TLS (Transport Layer Security). Požadavek je bezpečně poslán ke koncovému zařízení používáním bezpečnostních pravidel v síti. Šifrování média (RTP stream) je mimo rozsah SIP a musí být řešeno samostatně. Největší překážkou v protokolu SIP je přenos komunikace přes NAT, jak se můžeme dočíst v následujícím odstavci. [7]

„Jelikož SIP zabaluje adresní informace do svých datových rámců a NAT pracuje nejníže na síťové (network) vrstvě, adresní informace je automaticky změněna a média streamy nemají správnou adresní informaci potřebnou ke správnému připojení skrz NAT.“ [7]

Protokol SIP nespecifikuje konkrétní použití transportního protokolu a není také spojen s určitými komunikačními protokoly. Uvnitř zprávy protokolu SIP je proto zapouzdřena zpráva jiného protokolu, který nám určuje použité kódování pro multimediální data, jejich parametry a čísla portů, na kterých budou data vysílána nebo přijímána. Obvykle se o tuto operaci stará protokol SDP (Session Description Protocol). [7]

2.3 RTP protokol

RTP (Real Time Protocol) je protokol pro přenos dat v reálném čase. Je používán pro přenos audio a video provozu mezi koncovými procesy VoIP terminálů. „Protokol sám o sobě nezaručuje přenos dat v reálném čase, disponuje jen procedurami, které umožňují rekonstrukci těchto vlastností na straně přijímacího procesu.“ [1] V záhlaví protokolu se přenáší informace o typu přenášených dat, způsobu kódování, zdroji synchronizace, jakož i sekvenční číslo paketu a časová značka pro obnovu synchronizace a real-time vlastností. Zabezpečení interpretace dat z paketů RTP ve správném pořadí zajišťuje přijímací proces prostřednictvím sekvenčních čísel paketů. [1]

2.4 RTCP protocol

RTCP (Real Time Control Protocol) je řídicí protokol, který spolupracuje s protokolem RTP. Používá periodické vysílání paketů od každého účastníka relace RTP ke všem ostatním účastníkům a to za účelem řízení výkonnosti a pro diagnostické účely. [1]

RTCP vykonává následující služby: poskytuje informace aplikaci týkající se kvality vysílaných dat, identifikuje zdroj RTP, provádí řízení intervalu vysílání RTCP a přenáší minimální informace o řízení relace. [1]

2.5 Kodeky

Kodek je zařízení nebo algoritmus, který slouží k úspoře objemu přenášených dat. Díky kodekům umožňuje VoIP převod dat na hlas a jsou přenášeny v RTP protokolu. Kodeků existuje celá řada, liší se poměrem kvality a vyžadovaného pásma pro přenos. Kodeky G.729 a G.723.1 podporují potlačení ticha, což znamená, že se ticho během hovoru nepřenáší, neboť nenese žádnou informaci. [13]

V roce 1972 proběhla standardizace PCM neboli pulzní kódové modulace, která změnila oblast telekomunikací a nastartovala její digitalizaci. Digitalizace odsunula definitivně problémy analogové technologie a přinesla zrychlení spojovacího procesu, zvýšení spolehlivosti i dostupnosti služby. Ve VoIP se používají i další kódování, jako DPCM což je modifikací PCM, ADPCM, které vychází z DPCM a všechny zmíněné jsou založeny na principu vycházení z kvantifikace průběhu signálu. Můžeme se ještě setkat s kódováním LPC a jeho vylepšenou verzí CELP. Tyto způsoby kódování vychází ze znalostí o hlasovém traktu (tj. hlasivkách a krku). Nyní si popíšeme pouze ty nejzákladnější a nejvíce používané kodeky. [2]

G.711 – jedná se o základní kodek veřejné telefonní sítě, využívá modulační metodu PCM (pulzní kódová modulace), která převádí analogový zvukový signál na signál digitální. Využívá dvě kódovací metody a to **μlaw** v Severní Americe a **alaw** pro zbytek světa. Jeho přenosová rychlost činí 64 000 bitů za sekundu. Je základem pro další vyvinuté kodeky a je povinný pro všechna zařízení. Mezi jeho výhody patří minimální nároky na CPU. [13]

G.726 - tento kodek patří mezi kompresní kodeky a byl používaný v minulých letech. Nyní se s ním moc nesetkáme. Je známý pod názvem ADPCM (Adaptivní diferenční pulzní kódová modulace). G.726 má skoro identickou kvalitu jako G.711, ale jeho šířka pásma je poloviční. Je to dané z toho důvodu, že neposílá výsledek kvantizačního měření, ale posílá pouze informaci o popisu rozdílu mezi předchozím a současným vzorkem. [13]

G.729A – tento kodek používá CS-ACELP (Conjugate-Structure Algebraic–Code–Excited Linear Predictions) a díky němu doručí kvalitní zvuk při použití menší šířky pásma. CELP patří mezi populární metody komprese hlasu. „Je budován codebook zvuků matematickou modelací různých lidských hlasů. Místo toho, aby posílala aktuální vzorek hlasu, pošle odpovídající kód vzorku hlasu. Kvůli patentům však nemůžeme kodek volně použít, ale musíme platit za jeho licenci.“ [13]Má velkou podporu různých telefonů a systémů a je velmi oblíbený. Využívá hodně výpočetního výkonu CPU k dosáhnutí kompresního poměru. Využívá 8Kbps šířku pásma. [13]

GSM – má podobné parametry jako kodek G.729A, rozdíl ovšem spočívá v tom, že je zdarma dostupný a použitelný. Operuje na 13 Kbps. [13]

V následující tabulce 1 jsou uvedeny nejpoužívanější typy kodeků s jejich typem kódování, bitovou rychlostí, parametrem MOS, který vyjadřuje kvalitu (1-5) a výpočetní výkon.

Tabulka.1: Kodeky a jejich parametry 2.1

Standard	Algoritmus	CBR (kbps)	MOS	MIPS
G.711	PCM	64	4,1	---
G.723.1	ACELP	5,3	3,65	20
G.726	ADPCM	32	3,85	1
G.728	LD-CELP	16	3,61	30
G.729	CS-ACELP	31,2	3,92	20

3 PBX Asterisk

3.1 Co je to Asterisk

Asterisk (open source hybrid TDM and packet voice PBX) je softwarová ústředna, umožňující IP telefonii, digitální ISDN i analogovou telefonii. Jedná se o kompletní open-source softwarové PBX, běžící na platformách Unix a Linux, poskytující všechny vlastnosti PBX. Asterisk je zcela zdarma a pokud dokáže splnit požadované funkční a hardwarové nároky, je schopen nahradit komunikační systémy renomovaných výrobců. Navíc nabízí IVR (Interactive Voice Response) a ACD (Automatic Call Distribution). IVR je automatický systém, ovládaný většinou tónovou volbou nebo hlasem. Realizuje různé služby od jednoduchých jako je systém hlasové pošty, až po složitější jako jsou sdělení zůstatku na účtu, informace o službách, vystavení objednávky apod. ACD se stará o automatické rozdělování hovorů dle stanovených pravidel (číslo volajícího, definovaná schémata atd). [8]

Za vznikem Asterisku stojí Mark Spencer, tehdy čerstvý absolvent Auburn University v Alabamě, který se v roce 1999 rozhodl napsat Asterisk namísto zakoupení potřebné PBX. Mark Spencer založil firmu Digium, která stojí za vývojem Asterisku a jelikož se software Asterisk prodávat nedá, jelikož je volně šiřitelný, její profit plyne z technické podpory a prodeje hardwaru, který je plně kompatibilní s Asteriskem. Dnes je vývoj Asterisku především záležitostí open-source komunity a díky tomu má vývoj kolem 400 přispěvatelů, což je velká síla a velkou měrou se podílí na vývoji posledních verzí. Nejnovější verze Asterisku je 11 z roku 2012. [8]

Asterisk může být použit v těchto aplikacích: pobočková ústředna včetně rozhraní do PSTN, VoIP gateway pro různé protokoly (MGCP, SIP, IAX, H.323) a rozhraní, voicemail služby s adresářem, interaktivní hlasový průvodce (IVR), softswitch jako čistě softwarové řešení komunikačního serveru, konferenční server (funkce Meet me, konferenční místnosti), pro šifrování spojení, pro překlad čísel, pro systém předplacených volání, systém pro směrování cestou nejnižších nákladů (LCR), centrum volání (Call Center) a vzdálené kanceláře pro existující PBX, TDM přes Ethernet. [8]

3.2 Podporované technologie a hardware

„Systém je navržen tak, aby povoloval použití nových rozhraní a umožňoval snadno přidávat nové technologie. Jeho cílem je podpora veškerých možných typů současných i budoucích telefonních technologií. Obecně jsou rozhraní rozdělená do tří základních skupin:“ [9]

Dahdi hardware - tvorba nenákladných rozhraní nepředstavovala vůbec jednoduchý úkol. TDM hardware byl patentován a také byl příliš drahý. Nová myšlenka přišla s přidáním hostitelského procesoru, aby s ním mohl pracovat přímo Asterisk. CPU se postupem času stávaly rychlejšími a rychlejšími, proto bylo rozumnější pro tohle TDM zpracování ponechat software využívat hlavní CPU počítače. Po přidání TDM podpory začala firma Zapata Telephony vyrábět pseudo TDM rozhraní, které pojmenovala Zaptel. Architektura pseudo TDM poskytuje téměř stejnou kvalitu a real-time schopnosti jako hardware TDM. Její výhodou je podstatně nižší cena a vyšší flexibilita. Zaptel rozhraní dnes již pod názvem Dahdi dodává fima Digium pro různé síťové rozhraní jako PSTN, POTS, T1, E1, PRI, PRA a další. [9]

non-Zaptel hardware – tato rozhraní nepodporují pseudo-TDM komutování, ale poskytují konektivitu směrem k tradičním telefonním službám. Obsahují tyto typy rozhraní: ISDN4Linux, OSS/ALSA, Linux Telephony Interface, Phonejack/Linejack a Dialogic hardware. [9]

Packet voice – jde o standartní protokoly pro komunikaci přes paketové sítě (IP a Frame Relay). Tyto rozhraní jsou jediná, která nepožadují specializovaný hardware. Autor Asterisku Mark Spencer navrhl a zrealizoval svůj vlastní protokol IAX, který se stará o signalizaci a transport packet voice mezi dvěma připojenými uzly. IAX je schopen spojit každé dva koncové body podporující tento protokol. Následně byla přidána podpora pro další voice packet protokoly. Jde o protokoly SIP, H.323, MGCP a VoFR (Voice over Frame Relay). [9]

Asterisk je schopný komunikovat s širokou řadou různých technologií. Pro klasické připojení k telekomunikačním technologiím, jako je PSTN, vyžaduje specifický hardware v podobě karet. Karty telefonního rozhraní jsou PCI nebo PCI Express karty, které propojují počítač s Asteriskem přímo na vedení telefonů. Karty převádí signalizaci a média do interního formátu Asterisku. Nejznámější firmy, které vyrábějí tyto karty, jsou Digium a Sangoma. Pro práci s těmito kartami potřebuje balíček DAHDI (Digium Asterisk Hardware Device) obsahující ovladače pro tyto karty. [10], [11]

Karty se dělí na 3 typy:

- Analogové – podporují POTS, analogové linky a telefony. Používají samostatné moduly FXO a FXS.
- Digitální – slouží pro rozhraní ISDN a přípojky T1/E1 a PRI.
- Kompresní – využívá proces překódování, což je konverze kodeků v reálném čase. Překódování je náročné a tak využívá pomoci procesoru při kódování hovoru. [8]

3.3 Architekutura

Architektura je zcela odlišná od nejčastějších používaných telefonních produktů, ale je v podstatě velmi jednoduchá. Asterisk je v podstatě středový prvek spojující telefonní technologie s telefonními aplikacemi. Telefonní technologie mohou obsahovat jak VoIP služby (SIP, H.323, IAX atd.) tak i TDM technologie (T1, ISN, PRI, BRI atd.). Asterisk byl navržen za účelem poskytnutí maximální flexibility. Ústředna Asterisk je tvořena centrálním jádrem PBX, kolem kterého jsou specificky definovaná API. Toto jádro ovládá specifické protokoly, kodeky, hardware a díky tomu vykonává základní funkce jako je propojování hardwaru a aplikací. [9]

Jádro ovládá tyto položky:

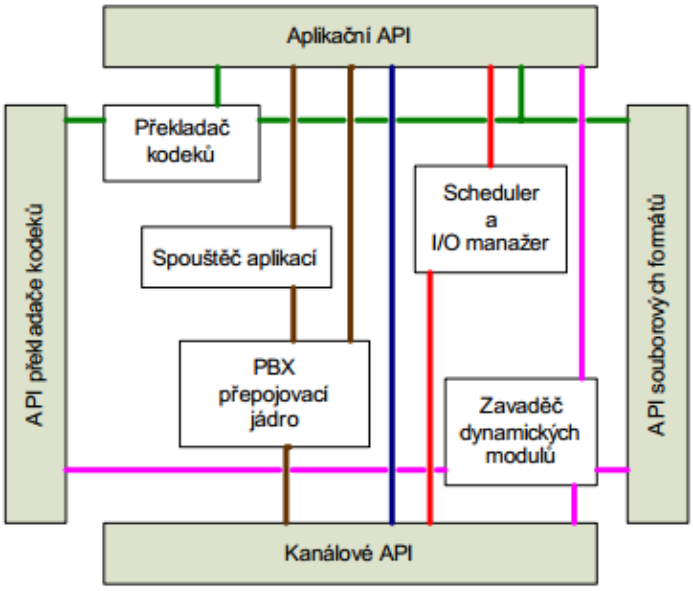
PBX přepojování (PBX Switching) – hlavním cílem Asterisku je propojování v PBX, spojování hovorů mezi různými uživateli a automatizovanými úlohami. Přepojovací jádro transparentně spojuje příchozí volání na různých HW a SW rozhraních.

Spouštěč aplikací (Application Launcher) – spouští aplikace zprostředkovávající služby jako jsou hlasová pošta, přehrání souboru nebo výpis adresáře.

Překladač kodeků (Codec Translator) – využívá moduly kodeků pro kódování/dekódování různých zvukových kompresích formátů používaných v telefonním prostředí. Existuje mnoho dostupných kodeků, které jsou vhodné pro různé potřeby a docílení rovnováhy mezi zvukovou kvalitou a využitou šířkou pásma.

Scheduler a I/O manager (Schedule and I/O manager) – ovládá nízko úrovňové úlohy a systémové řízení pro optimální výkon dle stavu zatížení. [9]

Na obrázku 3 vidíme blokové schéma architektury.



Obrázek 3: *Blokové schéma architektury 3.1 [8]*

3.4 Kanály

Jsou to logická spojení s různými signalizačními a přenosovými cestami, které Asterisk může využít k vytváření a spojování jednotlivých hovorů. Kanál může představovat spojení s obyčejným telefonním přístrojem nebo Internetové telefonní hovory. Asterisk nerozlišuje typy kanálu „FXO“ a „FXS“ a ani rozdíl mezi telefonními linkami a telefony. Každý hovor je umístěný na odlišném kanále. [9]

„Prostřednictvím kanálů vstupují do systému různé formáty komunikace; fyzické telekomunikační okruhy (jako FXO, FXS, PRI, BRI), softwarově založené spojení, síťově připojitelné entity (jako SIP a IAX) a exkluzivní vnitřní kanály Asterisku určené pro všechny dodatečné prostředky (jako Agent, Console, Local). Asterisk se všemi těmito kanály zachází jako s přípojnými body, jejichž vzájemnou interakci provádíte v Dialplan (extensions.conf).“[9]

Důležité je, že i když se kanály liší v rámci použité technologie nebo konektivity, Asterisk dovolí zacházet se všemi, jako by byly téměř stejné. Díky tomuto způsobu je Asterisk extrémně flexibilní a silnou PBX. Asterisk umožňuje jednat s daným kanálovým typem stejným způsobem jako s ostatními kanálovými typy. [9]

Ve standartní distribuci Asterisku se nachází následující typy kanálů: ACD Agent kanál, konzolový klientský ovladač pro zvukové karty, H.323 protokol, IAX nebo IAX2 protokoly, Local Loopback do dalšího kontextu, MGCP, Modem pro připojení ISDN linek, Linuxový telefonní kanál, SIP protokol, ovladač pro Cisco Skinny Client Control Protocol, Voice over Frame Relay, VPB pro připojení obyčejného telefonu a telefonní linky používající Voicetronix karty a Zap pro připojení obyčejného telefonu a telefonní linky používající Digium karty. [9]

3.5 Dialplán

Dialplán patří mezi nejdůležitější části Asterisku. Je to konfigurace, která nám určuje, jak má Asterisk zpracovávat hovory. Dialplán je konfigurován v souboru extensions.conf. Silnou stránkou Asterisku je, že tento soubor není závislý na technologii. Díky tomu tak lze Asterisk používat jako bránu mezi odlišnými technologiemi. Dialplán tvoří sekce general, globals, contexts a extensions. [9] [12]

V general můžeme nadefinovat obecné možnosti diaplánu např. uložení dialplánu, které bude pod hlavičkou general vypadat takhle:

```
[general]

static=yes
```

Sekce globals tvoří globální proměnné, které se používají jako konstanty. Příklad jak dlouho bude hovor vyzvánět, než se přepne do hlasové schránky:

```
[globals]

RINGTIME=>5
```

Dialplán se skládá ze souboru mnoha kontextů, přičemž kontext je souborem několika extensions. Kontexty jsou nejdůležitější části konfiguračního souboru i celé konfigurace systému. Mohou být používány za účelem implementování čísel důležitých vlastností jako například security pro povolení mezinárodních volání pouze z určitých telefonů, routing na směrování hovorů na základě extension nebo autoattendant pro uvítání volajících a vyzvání k volbě extension. Mezi další vlastnosti může patřit multilevel menus, ověření pravosti na přístupové heslo pro určitou pobočku, callback neboli redukce mezinárodních poplatků. Dále mohou zahrnovat také seznam nežádoucích volajících čísel na dané extension a PBX multihosting, který dokáže vytvářet „virtuální hosty“ na PBX. Poslední možné vlastnosti mohou být změna chování na základě hodin nebo vytváření skriptů pro běžně používané funkce. [9]

Při vytočení extensions, jsou příkazy provedeny podle jejich priorit (1, 2, 3…). Definice extensions obsahuje jeden nebo více příkazů. Každý příkaz je uveden na samostatném řádku v tomto formátu: [9]

```
exten => extension,priority,Command(parametrs)
```

Příklad kdy extensions 100 signalizuje volajícímu, že volaný telefon vyzvání pomocí příkazu Ringing. Dále nastane navázání hovoru příkazem Answer a následné přehrání zvukového záznamu za pomocí příkazu Playback. Poté se hovor ukončí příkazem Hangup. Velmi důležité je nezapomínat, že při každé změně dialplánu je potřeba provést v konzoli Asterisku příkaz dialplan reload, jinak by se změny neprojevily. [12]

```
exten => 100,1,Ringing()

exten => 100,2,Answer()

exten => 100,3,Playback(hello-world)

exten => 100,4,Hangup()
```

4 Autokonfigurační směrovací systém

Praktická část byla zaměřena na algoritmický návrh pro autokonfigurační směrovací systém pro SW PBX Asterisk a následné srovnání navrhovaného řešení s technologiemi ENUM a DUNDI. Nejdříve jsem provedl samotný algoritmický návrh a poté jsem se snažil o jeho simulaci. Byly mi poskytnuty 3 virtuální servery, na kterých jsem řešil simulaci tohoto autokonfiguračního systému.

4.1 Rozvržení SIP.conf

K dispozici jsem dostal tři virtuální servery v rozsahu IP 158.196.244.168-170. Tyto servery využívají operační systém Linux, na který jsem nainstaloval Asterisk. Na každém serveru jsem vytvořil novou verzi konfigu v souboru sip.conf. Vytvořil jsem v něm sekci general, která určuje společné nastavení pro všechny účty a přidal atribut udpbindaddr, ve kterém se vždy nachází IP adresa virtuálního stroje. Poté jsem vytvořil dva nové soubory sipppeers.conf a sipphones.conf, na které jsem odkázal v souboru sip.conf:

```
#include sipphones.conf
#include sippeers.conf
```

Soubor sipphones.conf slouží pro uložení informace o jednotlivých telefonech. V mém případě na IP adrese 158.196.244.168 jsou telefony 100 a 101. Více v ukázce kódu pro nastavení telefonu s číslem 100.

```
[100]
type=friend
context=dbtest
host=dynamic
disallow=all
allow=alaw
qualify=1000
secret=test
```

Sekce 100 je peer, který je dostupný pod tímto číslem. Obsahuje jednotlivé atributy:

- type=friend znamená, že lze použít pro příchozí i odchozí hovory
- context=dbtest je jméno kontextu obsažené v extensions.conf, které se bude provádět, když bude telefon realizovat odchozí hovor
- host=dynamic zajistí, že se telefon může k ústředně přihlásit z libovolné IP adresy
- disallow=all zakáže všechny typy kodeků
- allow=alaw povolí kodek G.711
- qualify=1000 je doba odezvy v milisekundách, pro kontrolu jestli je zařízení stále on-line
- secret=test je heslo které jsem zvolil jednoduché a je na všech přístrojích stejné

Konfigurace telefonu 101 se liší pouze v sekci peeru, který by byl dostupný pod číslem 101.

Soubor sippeers.conf je před začátkem spojení zcela prázdný. Po spojení PBX Asterisku se do něho zapisují informace o jednotlivých peerech pomocí definovaného vzoru.

Ukázka zápisu vzoru v sippeers.conf na PBX A, která se dotazovala na PBX B:

```
[158_196_244_169]
type=peer
defaultuser=158_196_244_168
secret=test
context=dbtest
disallow=all
allow=alaw
qualify=5000
host=158.196.244.169
```

Kde sekce peeru je vlastně zakódovaná IP adresa přijaté PBX. Obsahuje následující atributy:

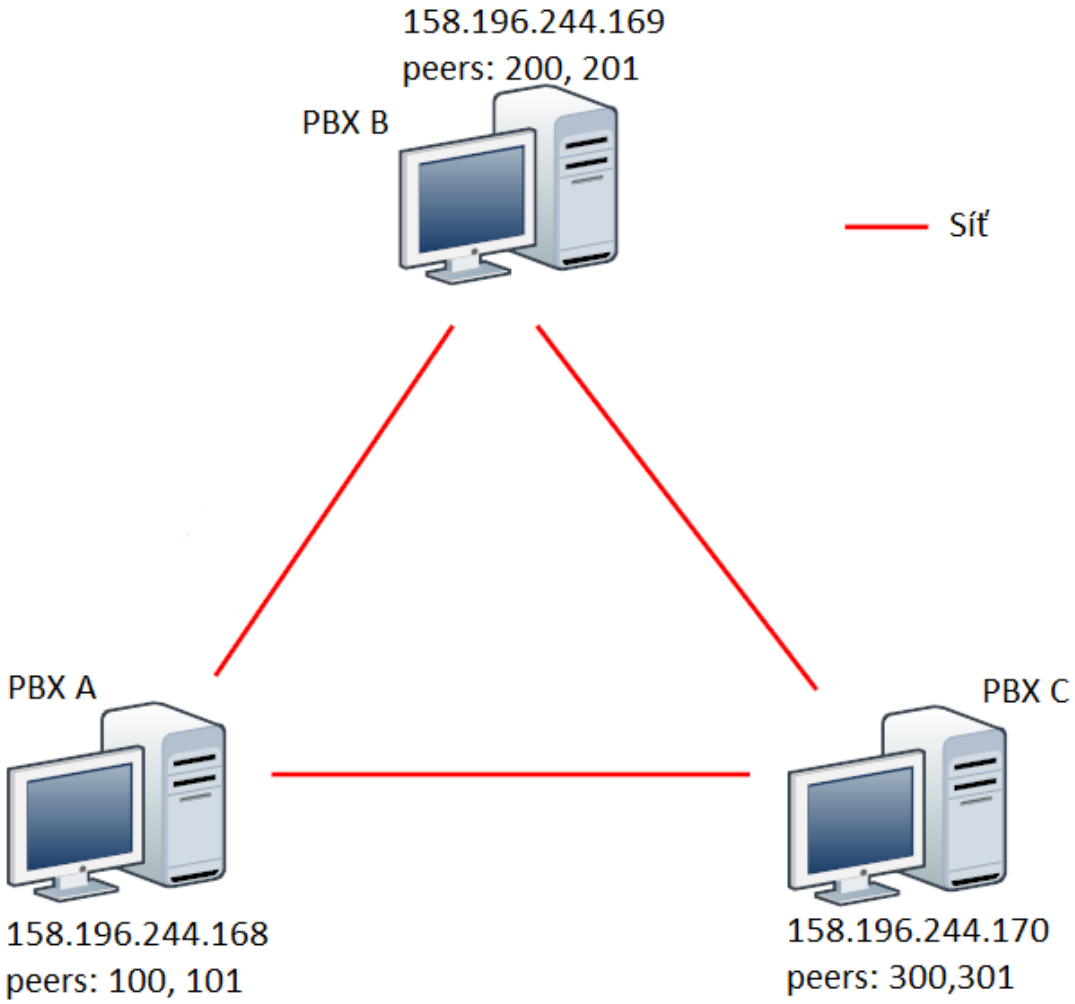
- type=peer znamená, že na daný uzel můžeme posílat hovory a můžeme je z něj přijímat, ale autorizuje se pouze na základě IP, nikoliv SIP uri (příp IAX2 URI)
- defaultuser=158_196_244_168 je nový název pro username
- host=158.196.244.169 zajistí, že se telefon může k ústředně přihlásit pouze z uložené IP adresy

Zbytek atributů je již popsán výše.

Ve vzoru se prakticky mění jen 3 části. Sekce, která představuje zakódovanou přijatou IP adresu, atribut defaultuser obsahující název pro username, jako zakódovanou lokální IP adresu a poslední atribut host, kde se nachází IP adresa přijaté PBX ve správném tvaru.

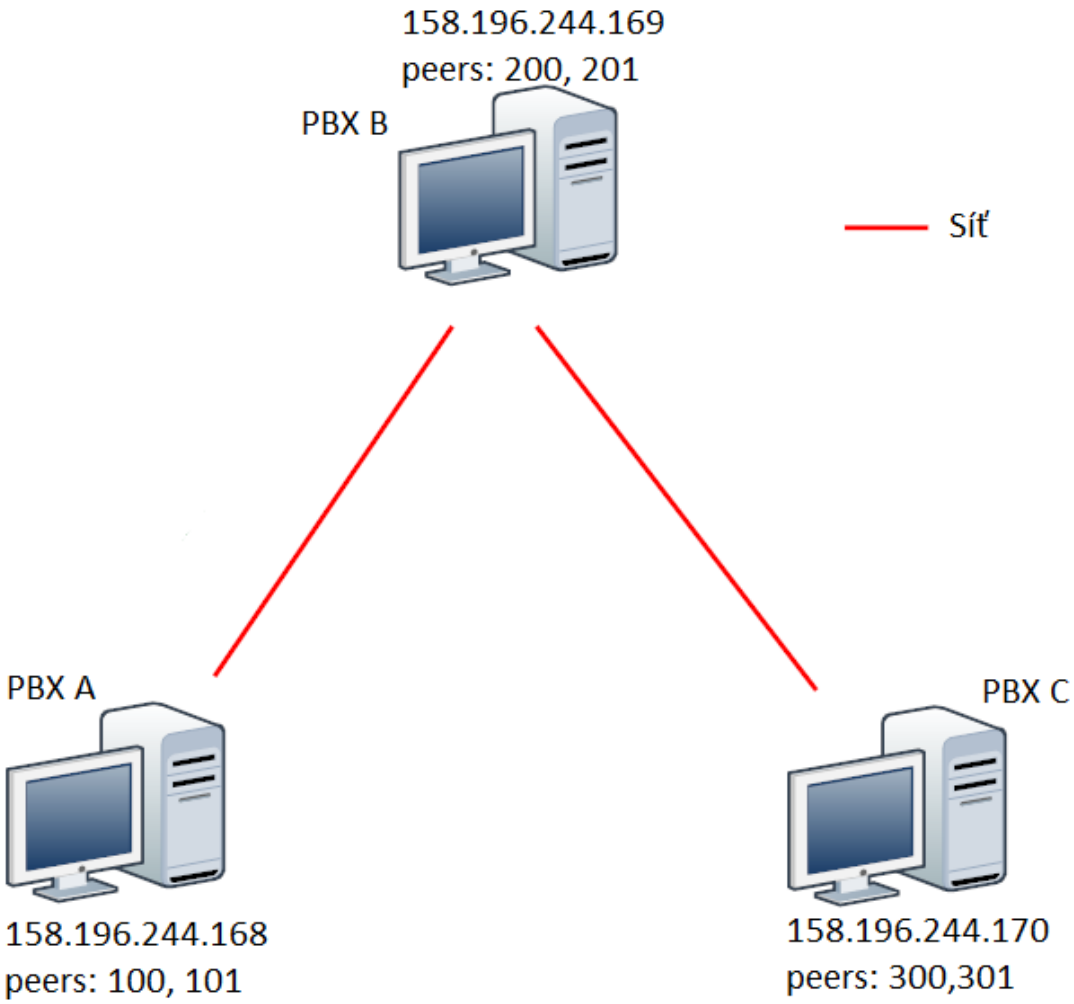
4.2 Experimentální topologie

PBX mohou být zapojeny v trojúhelníkové topologii, kde se jednotlivá PBX dotazuje na své sousední PBX, aby spolu mohli komunikovat. Topologie je zobrazena na následujícím obrázku 4.



Obrázek 4: *Trojúhelníková topologie 4.1*

Při návrhu jsem počítal i s typem topologie, kde se jednotlivé PBX budou dotazovat přes ostatní, aby spolu mohli komunikovat. Tento problém jsem vyřešil pomocí databáze Asterisku, kde jsem zapisoval jednotlivé délky kroku k dané pobočce. Navíc jsem musel ošetřit, aby v databázi byla uložena vždy nejkratší cesta k dané pobočce. Například při spojení PBX A s PBX B si vymění a následně uloží předané informace s krokem 1 do databáze. Nyní spolu mohou tyto dvě PBX komunikovat. Následně se spojí PBX C s PBX B a uloží si přijaté informace. PBX C si uloží informace o PBX B s krokem 1 a jelikož PBX B již zná informace od PBX A, PBX C si je uloží také ale s krokem 2. Nyní se může PBX C dovolat na PBX B i PBX A. Na následujícím obrázku 5 je ukázka této topologie.



Obrázek 5: *Ukázka topologie při využití počtu kroků v databázi 4.2*

5 Návrh autokonfiguračního směrovacího systému

Pro vytvoření propojení trunkovým spojem dvou a vícero Asterisků, je potřeba si vyměnit informace, které jsou uloženy v souboru sip.conf. Alternativou je předávání IP adresy PBX a přihlašovacích údajů dynamicky, jako to řeší například systém DUNDi. Z hlediska komplexnosti je dobré tento problém rozdělit na několik logických funkcí, které se dají snáze spravovat. Logicky lze strukturu rozdělit do tří funkcí – předávání lokálních informací potencionálním peerům, aplikace přijatých směrovacích informací a kontrola konzistence databáze směrovacích informací. Tento návrh je taky výhodný z hlediska, že po spojení všech PBX obsahuje každá PBX kompletní databázi všech Asterisků. Může se spojit s kteroukoliv pobočkou.

Prvním problémem je zjištění IP adres spuštěných PBX, tedy takových s otevřeným portem. Pro poslání informací musí mít daná IP adresa otevřený port, pro přijmutí posílaných informací. Na každé PBX se musí nacházet soubor, ve kterém je uložený rozsah sítě nebo jednotlivé IP adresy PBX. Z něho se postupně vyberou všechny IP adresy a zjistí, zda mají otevřený port. Otevřený port na dané IP adrese znamená, že je PBX schopna na této adrese přijímat informace. Jestliže nenajde žádnou IP adresu s otevřeným portem, zjišťuje znova, zdali nějaká IP adresa nově neotevřela port. Je to z důvodu, že při žádné IP adrese s otevřeným portem, by se neměli dané informace kam posílat.

Při problému odesílání směrovacích informacích, se bavíme o přenosu IP adresy PBX spolu s jednotlivými čísly peeru a jejich hodnotami pro počet kroků. IP adresa je přenášena, aby bylo zřejmé, kde se přesně PBX nachází a čísla jednotlivých peerů pro seznam poboček na PBX. Hodnota počet kroků určuje délku cesty k pobočce. Díky této hodnotě se přesně ví, přes kolik PBX musí projít, než se spojí s danou pobočkou. Pro takovýto návrh odesílání dat, je nutno upravit soubor sip.conf, jak je popsáno výše. Po úpravě souboru, obsahuje pouze potřebné informace v přehlednější formě, pro snazší zpracování. Čísla ze souboru, kde jsou uloženy informace o jednotlivých telefonech, se budou zapisovat do databáze Asterisku, pro lepší přehlednost, pro práci s dialplánem a možnost dynamické úpravy bez nutnosti reloadovat Asterisk. Ke každému číslu v databázi se navíc zapíšou hodnoty host, step a timestamp. Názvy hodnot jsou určeny z anglických slov, jejichž název naznačuje, co se zde bude ukládat. Hodnota host se zapisuje pomocí IP adresy. Pokud bude číslo z dané PBX, zapíše svou IP adresu unikátním slovem local. Takhle snadno po nahlédnutí do databáze, ihned poznáme lokální čísla dané PBX. Další hodnota, která se bude zapisovat spolu s číslem, je step neboli krok. Tato hodnota určuje počet kroků k dané pobočce. Musí být ošetřena, aby se v databázi Asterisku nacházela vždy nejkratší cesta k dané pobočce. Obecně, proč si natahovat cestu, když známe kratší. Může zde nastat ještě problém při vymazání čísla ze souboru, kde jsou uloženy informace o jednotlivých telefonech. Toto číslo v databázi by se pak nacházelo zbytečně, jelikož by se s ním nedalo navázat spojení. Pro takovýto případ by bylo využito zapsání speciálního kroku do databáze Asterisku. Pokud se odpojí nějaké číslo, v databázi Asterisku dojde k přepsání hodnoty krok speciálním krokem. Speciální krok by obsahoval číslo 9001. Číslo je zvoleno záměrně vysoké, aby se lišilo od ostatních kroků. Takovéto čísla se speciálním krokem, by později navrhnutá kontrola informací v databázi vymazala. Poslední hodnota, která se bude ukládat s číslem je timestamp. Jedná se o zapisování aktuálního času do databáze. Chceme mít v databázi Asterisku vždy aktuální data. Pomocí této hodnoty se může databáze Asterisku mazat v určitém časovém intervalu. Tím se zajistí, že se v databázi budou nacházet stále aktuální data. Vytvoří se posílaná směrovací informace ve formátu JSON, která bude obsahovat IP adresu PBX a její čísla s hodnotou step. Formát JSON, protože je jednoduše čitelný i zapisovatelný člověkem a snadno analyzovatelný i generovatelný strojem. Navíc je velmi vhodný k serializaci strukturovaných dat (slovníky, pole, atd..).

Informace může vypadat takhle:

{“IP”: {“158.196.244.168”}, “numbers”: {“100”: ”0”, “101”: “0”}}

Může nastat problém při neúspěšném přenosu těchto informací, například pří výpadku sítě nebo pádu Asterisku. Tento problém by se mohl řešit pomocí 5 pokusů, které se mohou po časové prodlevě znovu pokusit o přenos informací na danou PBX. Pokud se přenos informací nezdaří ani po zmíněných 5 pokusech, systém vypíše upozornění o nezdařilém přenosu informací pro danou PBX. Tento problém může vzniknout například, kdy daná PBX nečekaně uzavře svůj port.

Řešíme-li problém přijatých směrovacích informací, pak se bavíme zejména o přijatých informací z dané PBX, které si chceme uložit do svojí databáze Asterisku. Pro přijmutí informací, musí PBX naslouchat na určitém otevřeném portu. Systém přijaté informace kontroluje, zda jsou informace v již zmiňovaném formátu JSON. Pokud nastane problém ve formátu přijatých dat, tyto informace jsou neplatné a dále se s nimi nepracuje.

Po přijmutí informací ve správném formátu, nastává problém v zapsání peeru do již vytvořeného souboru, kde se zapisují jednotlivé peery. Do tohoto souboru je zapisován peer podle určeného vzoru. Tento vzor je rozepsán výše. Vzor je zde vytvořen, proto jelikož se v něm mění pouze 3 údaje. Jedná se o název peeru, který se zapisuje pomocí zakódované IP adresy PBX, z které byly odeslány informace. Atribut username, do kterého se ukládá pro změnu zakódovaná IP adresa PBX, na kterou se poslali informace. Posledním údajem je atribut host, ve kterém je zapsaná IP adresa PBX, z které byly odeslány informace. Zbytek údajů se nemění a jsou zapsány ve výše zmíněném vzoru. Systém zjistí, zda pro přijatou IP adresu je peer znám. Pokud ne uloží ho. Pomocí tohoto peeru se systém přepojuje na PBX, která obsahuje volanou pobočku.

Po zapsání peerů nastává problém zapsání informací o telefonech. Přijaté informace jsou ukládány do databáze Asterisku. Tyto čísla se zapisují opět s hodnotami host, step a timestamp. Může se nastat několik problému, které dále rozepíšu.

Přijaté číslo již bude zapsáno v databázi Asterisku s hodnotou host local. To nám vlastně říká, že dané číslo pochází z této PBX a není nutno ho přepisovat. Tímto je vyřešeno nepřepisování lokálních poboček, jelikož vždy obsahují nejkratší cestu, protože se nachází na dané PBX.

Následující problém může být, že pro přijaté číslo není v databázi Asterisku IP adresa. V takovém případě se do databáze Asterisku zapíše číslo se všemi hodnotami (host, step a timestamp). PBX neznala údaje o tomto čísle, a proto byly všechny údaje s číslem zapsány do databáze Asterisku. Hodnota step se při zápisu na přijatou PBX vždy zvětší o 1 krok. To je dáno tím, že vždy urazí jeden krok z PBX, z které byly informace odeslány na PBX, kde se informace poslali.

Nastane-li problém, že pro přijaté číslo hodnota step není v databázi Asterisku, systém by to řešil pomocí vytvoření virtuální kroku, do kterého by uložil vysoké číslo kroku. Vytvořený virtuální krok tak zajistí porovnání počtu kroků s přijatým číslem. Číslo je použito vysoké záměrně, aby mohlo dojít k přepsání kratší cesty. Tímto je zajištěno přepsání počtu kroků pro číslo, kde se hodnota step nenachází v databázi Asterisku. Jeli naopak počet kroků znám, dojde ke klasickému porovnání délky cesty. Pokud má přijaté číslo menší hodnotu step, než hodnota step uložená v databázi, systém zapíše přijatou hodnotu step o 1 krok větší, kvůli uražené cestě z PBX, z které byly informace odeslány. Zároveň je přepsána i hodnota timestamp, na právě aktuální čas zapsání do databáze. Pokud by systém vyhodnotil, že přijaté číslo obsahuje větší počet kroků, než je počet kroků v databázi, nastává poslední problém. Přijaté číslo by mohlo obsahovat speciální krok. Systém ověří, zda obsahuje přijaté číslo speciální krok, což je krok s hodnotou větší než 9000. Toto číslo se posílá se speciálním krokem na všechny PBX, aby nevznikala nekonečná smyčka pro již odpojené číslo. Pokud obsahuje číslo

speciální krok, přepíše hodnotu krok a zároveň čas zapsání do databáze. Těmto číslům se speciálním krokem, bude později navrhnutá kontrola informací v databázi vymazávat. V případě, že neobsahuje speciální krok a zároveň pro přijaté číslo je počet kroků větší, než v databázi systém informace do databáze nezapisuje. Takto systém ošetřuje, aby se v databázi Asterisku nacházeli vždy nejkratší cesty k jednotlivým pobočkám.

Po změně v souboru sip.conf, respektive v tomto řešení se jedná přesně o soubor, kde se zapisují informace o jednotlivých peerech, je nutno aktualizovat nastavení v tomto souboru. Tento problém vyřeší v Asterisku příkaz:

```
sip reload
```

Tímto se aktualizuje nastavení SIPu. Asterisk bude pracovat již s provedenými změnami v konfiguračním souboru sip.conf.

Při řešení problému pro kontrolu směrovacích informací v databázi, se bavíme o tom, aby se v databázi Asterisku nacházela vždy aktuální data. Řešením tohoto problému jak již bylo nastíněno dříve, je speciální krok. Pokud by číslo, bylo odpojeno, bude obsahovat speciální krok. Kontrola databáze bude mazat všechny čísla se speciálním krokem. Tím je vyřešeno, že se v databázích Asterisku nebudou nacházet již nedostupná čísla. Může ovšem nastat problém při pádu Asterisku. V tomhle případě by neměl kdo PBX poslat informaci o čísle se speciálním krokem. Tento problém je možno řešit, pomocí mazání všech údajů z databáze Asterisku v pravidelném časovém intervalu. Systém by si tak uložil aktuální čas a jednotlivé časy pro daná čísla v databázi. Po rozdílu aktuálního času a času v databázi, který bude větší než nastavený pravidelný časový interval, by vymazal informace v databázi pro dané číslo. Tím je zcela zajištěno, že databáze Asterisku bude obsahovat pouze aktuální informace.

Nyní jsou navržené tyto 3 logické funkce - předávání lokálních informací potencionálním peerům, aplikace přijatých směrovacích informací a kontrola konzistence databáze směrovacích informací. Logicky může každá funkce představovat jeden skript, který bude řešit dané problémy. Tyto skripty pak můžeme nazvat obecně jako klienta, server a hlídače. Každá PBX musí obsahovat všechny tyto 3 skripty pro běh navrženého autokonfiguračního systému. Všechny skripty musí být spuštěny jako tzv. daemon (démon). Démon je vlastně skript, který je spuštěn dlouhodobě a není v přímém kontaktu s uživatelem. Jeho úkol spočívá ve vyčkávání na nějakou událost, kterou posléze obslouží. Tímto zajišťuje různé úkoly, bez nutnosti spolupráce s uživatelem.

Posledním problémem byla integrace vyměňovaných informací do aktivního dialplánu. Dialplán je obsažen v souboru extensions.conf a určuje jak má Asterisk zpracovávat hovory. Dialplán je nutno nastavit tak, aby bral jednotlivé informace pro volané číslo z databáze Asterisku. Více tento dialplán rozeberu při samostatné simulaci.

6 Simulace autokonfiguračního směrovacího systému

Po navrhnutém řešení jsem se snažil o jeho simulaci. Pro testování jsem měl k dispozici tři virtuální servery. Díky tomu jsem vytvořil na každém serveru soubor nets.conf a v něm nastavil na pevno zbývající 2 adresy pro komunikaci. Vytvořil jsem dva hlavní skripty server.sh a klient.sh v jazyce bash díky snadné simulaci daemona. Ostatní skripty jsem vytvářel v jazyce python, protože se snadno vkládá do jiných aplikací a kód programu je ve srovnání s jinými jazyky kratší a dobře čitelný. Pro vypsání jednotlivých IP adres z tohoto souboru jsem naprogramoval skript parse.py. Pro simulaci jsem vytvářel klienta na PBX A a server na PBX B.

Jako první jsem vytvořil skript server.sh v jazyce bash, kde jsem do nekonečného cyklu vložil poslouchání na zvoleném portu 10000 a pomocí funkce echo vypisoval přijaté informace z klienta. Tohle mi zaručilo, že server bude neustále poslouchat na mém zvoleném portu 10000 a vypisovat přijaté informace ze strany klienta. Dále jsem se pustil do vytváření skriptu klient.sh psaném také v jazyku bash. Vytvořil jsem nekonečnou smyčku a v ní seznam activ_ip. Do tohoto seznamu se uloží všechny IP adresy z nets.conf s otevřeným portem 10000. Následuje for cyklus, který prochází jednotlivé ip adresy ze seznamu activ_ip na které bude klient posílat dané informace. Vytvořil jsem proměnné process_status="1" a tries="5", abych ověřil, zda přenos byl úspěšný (pokud se proces provede v jazyce bash se vypíše hodnota "0") a pokud ne může se pětkrát zopakovat. Následuje smyčka while s podmínkou pokud se process_status ! = 0 nebo pokusy! = 0, pak provede zaslání informací, které vytváří skript sendip.py. Skript sendip.py obsahuje funkce getInfo(), pomocí které s určitým argumentem načteme IP adresu PBX a jednotlivé peery, dbread() s argumentem step pro načtení tel. čísla s aktuálním krokem z databáze a dbwrite() pro zapsání tel. čísel s hodnotou IP, počtem kroků a časem zápisu do databáze, které se určují podle argumentu. Následně načte čísla jednotlivých peeru ze souboru sipphones.conf. Tyto čísla uloží do databáze Asterisku s IP adresou local, počtem kroků 0 a aktuálním časem zapsání do databáze Asterisku. IP adresa se na klientovi zapisuje s určitým slovem local, jelikož se IP adresa rovná adrese, na které se Asterisk nachází a tudíž nemusí provádět žádné kroky pro spojení s telefony na této PBX. Následně jsem vytvořil slovník, do kterého jsem vložil informace s IP adresou, tel. čísly a jejich počty kroků. Tyto informace jsem získal pomocí výše uvedených funkcí. Slovník poté převedl do formátu JSON, a tuto informaci poslal pro právě načtenou IP adresu ze seznamu activ_ip. Pokud přenos proběhne úspěšně do proměnné process_status se dosadí 0, čili přenos proběhl v pořádku a while cyklus se znovu neprovede. Naopak při neúspěšném přenosu informací zůstává process_status=1 a proměnná tries se zmenší o 1. Tady je nastavena sekundová prodleva před dalším pokusem pro přenos informací. Jestliže se přenos informací nezdaří ani po 5 pokusech, je tento přenos neúspěšný. Po úspěšném či neúspěšném přenosu, načte další IP adresu z proměnné active_ip, na kterou bude posílat dané informace. Celý proces přenosu opakuje pro nově načtenou IP adresu. Pokud se již v active_ip nenachází další IP adresa, následuje 20 sekundová prodleva, po které se vrací se do počáteční nekonečné smyčky, kde začíná celý proces od začátku.

Klient už posílá dané informace na stranu serveru, který je přijme, ale neví co s nimi. Vytvořil jsem skript clientdata.py, který bude s těmito informacemi pracovat a vnořil ho do skriptu server.sh. Skript clientdata.py obsahuje funkce getInfo() na zjištěný IP PBX a čísla jednotlivých peerů podle určitého argumentu, dbwrite() a dbread() pro zápis a načtení informací v databázi s jednotlivými argumenty. Velmi důležitou funkcí je dbwritetemp(), která zapisuje peer do souboru sippeers.conf podle daného vzoru. Poslední funkce sipreload() slouží pro reloadnutí nastavení SIP. Skript clientdata.py ověří, zda přijatá data od klienta jsou ve formátu JSON. Pokud ne, vypíše upozornění, že přijatá data nejsou ve formátu JSON a dále s nimi už nebude pracovat, jinak je převede do slovníku.

Přijatou IP adresu zakóduje určitou metodou, která změní v IP adrese znak tečky za znak podtržítka a zeptá se, zda není obsažena jako peer v sippeers.conf. Pokud ne, zapíše ji pomocí výše zmínění funkce dbwritetemp() podle určeného vzoru, jinak přeskočí toto zapsání. Následně začne procházet jednotlivá přijatá čísla. Načte první přijaté číslo a následuje řada podmínek.

- Je pro přijaté číslo v databázi IP adresa zapsána jako local?
- Je pro přijaté číslo zapsána IP adresa v databázi?
- Je pro přijaté číslo hodnota počet kroků None?
- Je pro přijaté číslo počet kroků v databázi větší než počet přijatých kroků?

První podmínka ošetřuje, jestli není přijaté číslo zapsáno v databázi s lokální IP adresou. Tímto je zajištěno, nepřepisování lokálních čísel, které vždy zapisuje pouze klient. Druhá podmínka ověřuje, jestli je pro přijaté číslo zapsána IP adresa v databázi. Pokud se IP adresa pro číslo nenachází v databázi, může rovnou zapsat telefonní číslo s IP adresou, počtem kroků a aktuálním časem zápisu do databáze. V opačném případě pokud je číslo obsaženo v databázi, musí projít k následující podmínce, která zjišťuje, zda není pro přijaté číslo počet kroků prázdná hodnota v databázi. Pokud ano, vytvoří se proměnná krok, do které uloží vysoké číslo například 2000. Číslo 2000 z důvodu aby mohla být přepsána hodnota počet kroků kratší cestou. Pokud počet kroků není prázdná hodnota, uloží do proměnné krok aktuální počet kroků z databáze. Další podmínka ověřuje, jestli je hodnota proměnné krok větší než počet přijatých kroků. Pokud ano, zapíše do databáze počet přijatých kroků a čas zápisu do databáze, jinak nezapisuje nic. Tímto je vlastně zajištěno, aby se v databázi nacházela vždy nejkratší cesty k ostatním pobočkám. Tenhle cyklus podmínek provede pro všechny přijaté čísla. Poté co projde všechna přijatá čísla se reloadne nastavení SIP a vrací do počáteční nekonečné smyčky, kde začíná celý proces znovu.

Pro zajištění aktuálních dat v databázi Asterisku jsem vytvořil pomocný skript watch.py. Tento skript obsahuje funkce dbread() a dbdel() pro čtení a mazání informací z databáze pomocí jednotlivých argumentů. Je zde opět volána nekonečná smyčka. V ní se pomocí funkce dbread() s argumentem ‘ts’ vypíšou všechna čísla z databáze se svým časem zápisu, které uloží do slovníku tsdict. Pomocí for cyklu začne postupně procházet jednotlivá čísla. Po načtení prvního čísla uloží čas jeho zápisu do databáze do proměnné datcas. Aktuální čas taktéž uloží do proměnné aktcas. Následně provede rozdíl mezi aktuálním časem v proměnné aktcas a časem z databáze v proměnné datcas. Rozdíl mezi těmito časy uloží do proměnné rozdil. Čas v proměnné rozdil převede na sekundy. Následuje podmínka, pokud je čas v proměnné rozdil větší než 300 sekund, vymaže v databázi pro načtené číslo IP adresu, počet kroků a čas zápisu do databáze jinak nemaže nic. Tento cyklus provede pro všechny čísla ze slovníku tsdict. Následuje 60 sekundová prodleva, po které se vrací do počáteční nekonečné smyčky, kde začíná celý proces znovu. V simulaci, při odpojení pobočky na některé PBX, vznikne nekonečná smyčka pro tuto pobočku. Hlídač sice pobočku na dané PBX smaže, ale jelikož již informace o této pobočce poslal jiné PBX, vznikne pro tuto pobočku nekonečná smyčka. V databázi Asterisku se mohou vyskytovat již smazané pobočky, na které se nedovoláme. Možnost jak toto ošetřit je varianta s posíláním speciálního kroku, který jsem uvedl v návrhu řešení.

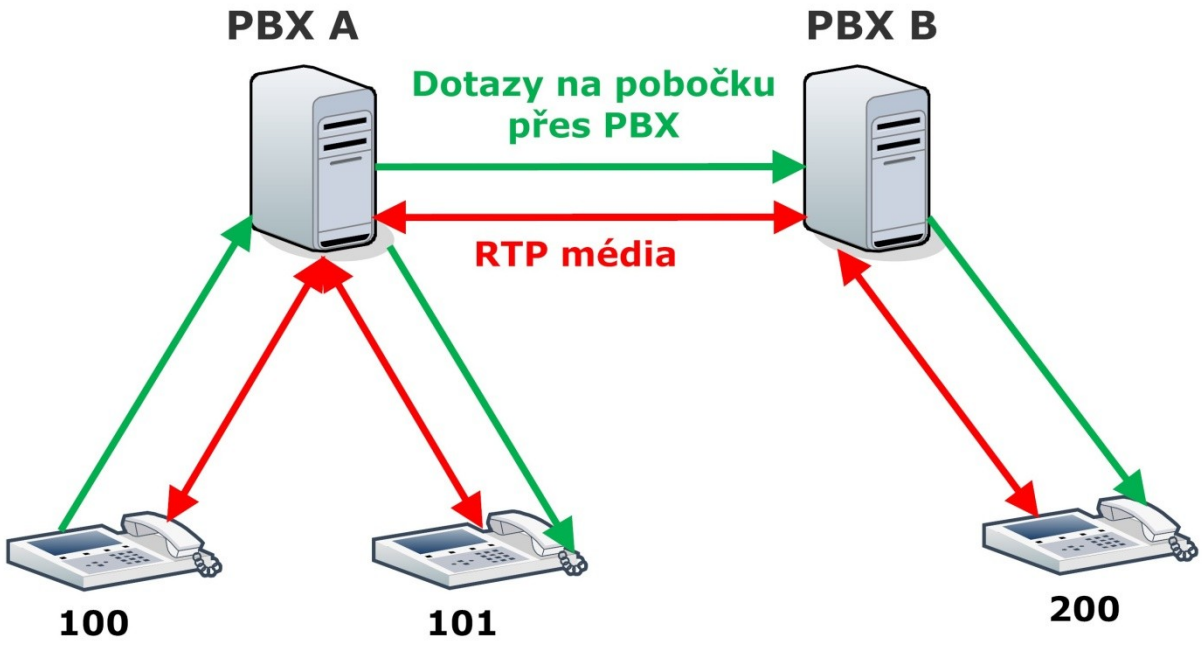
Každá PBX musí obsahovat 3 hlavní skripty server, klienta a hlídače.

6.1 Aktivní dialplán

Posledním krokem je upravit dialplán, který určuje jak má Asterisk zpracovávat hovory a nachází se v souboru extensions.conf. Zde je ukázka dialplánu dbtest.

```
[dbtest]

exten => _X.,1,Answer()

exten => _X.,n,GotoIf($["${DB(routing/host/${EXTEN})}" =
"local"]?localdial:remotedial)

exten => _X.,n(localdial),Dial(SIP/${EXTEN})

exten => _X.,n,Goto(end)

exten => _X.,n(remotedial),GotoIf($["${DB(routing/host/${EXTEN})}" =
""]?end)

exten => _X.,n,Dial(SIP/${DB(routing/host/${EXTEN})}/${EXTEN})

exten => _X.,n(end),Hangup()
```

V prvním kroku se spojí dotaz s PBX Asterisku. Následuje podmínka, jestli je vytočené číslo obsaženo v databázi Asterisku. Pokud ano, pošle dotaz pro vytočené číslo a skočí na prioritu localdial. Nyní dojde k vytvoření spojení pomocí příkazu Dial (), který se spojí s volanou extension a zahájí hovor s umožněním přenosu RTP dat. Následuje, skok na prioritu end. Zde se hovor ukončí příkazem Hangup (). Pokud se vytočené číslo nenachází v databázi Asterisku, pošle dotaz na volanou PBX Asterisku a skočí na prioritu remotedial. Na volané PBX se podívá do databáze Asterisku zda se zde nachází volané číslo. Pokud ano, vytvoří se spojení hovoru pomocí příkazu Dial (), který se spojí s volanou extension a zahájí hovor s umožněním přenosu RTP dat. Poté se hovor ukončí příkazem Hangup (). Jestliže se volané číslo nenachází ani na volané PBX Asterisku, přeskočí na prioritu end a hovor se neuskuteční.



Obrázek 6: *Ukázka hovoru pomocí dialplánu 6.1*

Na předcházejícím obrázku 6 je naznačeno fungování tohoto dialplánu. Je zde zobrazena ukázka, když by chtěla pobočka 100 volat na pobočku 101 nebo 200. Zelenou šipkou jsou označeny dotazy pro volanou pobočkou na jednotlivé PBX. Pokud PBX znají tuto pobočku, pošlou ji dotaz o žádosti spojení. Pobočka se spojí přes danou PBX s volanou pobočku a zahájí hovor pomocí přenosu RTP dat.

6.2 Ukázka z testování

Tímto je vše připraveno a stačí otestovat, zda se provede hovor mezi dvěma pobočkami na různých PBX. Na obrázku 7 jsou ukázané připojené telefony pomocí příkazu v Asterisku:

```
sip show peers
```

```
ub*CLI> sip show peers
Name/username             Host                                    Dyn Forcerport ACL Port     Status
100/100                   158.196.244.206                          D   a             5060     OK (36 ms)
101                       (Unspecified)                            D   a             0        UNKNOWN
158_196_244_169/158_196_2 158.196.244.169                              a             5060     OK (4 ms)
3 sip peers [Monitored: 2 online, 1 offline Unmonitored: 0 online, 0 offline]
```

Obrázek 7: *Ukázka peerů v Asterisku 6.2*

Na obrázku 8 je zaznamenám průběh vykonávání jednotlivých aplikací dialplánu v konzoli Asterisku. Hovor byl prováděn z PBX A na PBX B na volanou pobočku 200.

```
== Using SIP RTP CoS mark 5
  -- Executing [200@dbtest:1] Answer("SIP/158_196_244_168-00000032", "") in new stack
  -- Executing [200@dbtest:2] GotoIf("SIP/158_196_244_168-00000032", "1?localdial:remotedial") in new stack
  -- Goto (dbtest,200,3)
  -- Executing [200@dbtest:3] Dial("SIP/158_196_244_168-00000032", "SIP/200") in new stack
== Using SIP RTP CoS mark 5
  -- Called SIP/200
  -- SIP/158_196_244_168-00000032 requested media update control 20, passing it to SIP/200-00000033
  -- SIP/200-00000033 is ringing
  -- SIP/158_196_244_168-00000032 requested media update control 20, passing it to SIP/200-00000033
  -- SIP/158_196_244_168-00000032 requested media update control 26, passing it to SIP/200-00000033
  -- SIP/200-00000033 answered SIP/158_196_244_168-00000032
  -- Remotely bridging SIP/158_196_244_168-00000032 and SIP/200-00000033
== Spawn extension (dbtest, 200, 3) exited non-zero on 'SIP/158_196_244_168-00000032'
```

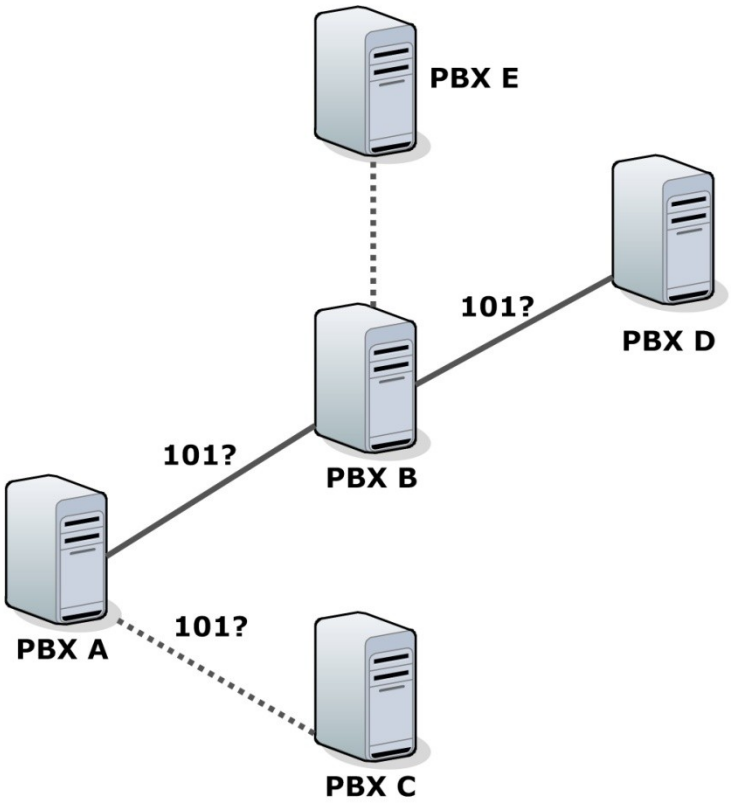
Obrázek 8: *Ukázka hovoru v Asterisku 6.3*

Mnou nasimulované řešení funguje. Hovor mezi dvěma pobočkami na různých PBX se uskutečnil bez problému.

7 Srovnání navrhovaného řešení s technologiemi ENUM a DUNDI

Nejprve si povíme něco o technologiích ENUM a DUNDI, které posléze srovnáme s navrhovaným řešením. Systém ENUM je jednoduše řečeno tzv. internetová databáze, kde majitelé telefonních čísel zveřejňují způsob, jak se jim mohou ostatní uživatelé na toto číslo dovolat přes internet. Pokud se volající dokáže podívat do této tzv. internetové databáze, jeho hovor může být spojen přes internet. Velkou výhodou je, že nezáleží, zda volá z pevné či mobilní sítě v České republice nebo ze zahraničí. ENUM funguje jako doplňková služba pro přenos hlasu po internetu, jako například VoIP služby telekomunikačního operátora, VoIP infrastruktura ve firmě nebo některá volně dostupná VoIP služba založená na standardu SIP. S pomocí ENUM může uživatel propojit svou VoIP službu s běžným telefonním číslem. Hovor pak probíhá běžným způsobem bez nutnosti znát uživatelské jména či adresy. Při využití VoIP služby, je hovor směrován k volanému účastníkovi částečně nebo zcela přes internet, kde se za objem přenesených dat neplatí. Systém ENUM právě eliminuje přenos přes běžnou telekomunikační síť a přenáší hovor přes internet celou dobu. Tímto jsou tyto hovory úplně zdarma. ENUM spojí telefonní číslo s internetovou adresou, pod kterou je dosažitelný telefonní přístroj s daným telefonním číslem. K tomuto využívá ENUM systém doménových jmen (DNS), který je v internetu obvykle uplatněn pro přiřazení doménových jmen k IP adresám. Vytvoří se speciální doménové jméno z telefonního čísla, které si uživatel zaregistruje a přidá do DNS adresu VoIP telefonu ve formě SIP adresy. Po obdržení požadavku na spojení s příslušným telefonním číslem, provede dotaz do DNS, zda ke konkrétnímu číslu existuje doména v systému ENUM. Pokud ano, systém vrátí SIP adresu konkrétního uživatele. Tímto se pro směrování hovoru použije tato adresa místo telefonního čísla. Tím je docíleno, že hovor směřuje pouze přes internet. Pokud k danému číslu neexistuje doména v systému ENUM, nezbude nic jiného než hovor spojit běžným způsobem přes veřejnou síť (za poplatky). Mezi výhody ENUM patří volání zdarma všem účastníkům registrovaným v ENUM, vysoká úspora za volání mezi pobočkami v ostatních zemích, nízké náklady na zavedení ENUM nebo možnost volat s ENUM pomocí pevného i mobilního IP telefonu. Naopak nevýhodou je, pokud není účastník registrovaný v ENUM, musí být použitá veřejná síť, kde se za hovor platí poplatky. Jeho využití je převážně ve velkých firmách, kde spolu zaměstnanci komunikují prostřednictvím této technologie. [14]

Nyní si povíme něco o technologie DUNDi. Jedná se o peer to peer systém pro lokalizaci internetových bran na telefonních službách. Na rozdíl od tradičních centralizovaných služeb, je DUNDi plně distribuované bez centrální autority. DUNDi je jako velký telefonní seznam, který umožňuje zeptat se kolegy, jestli nezná VoIP cestu s extensions číslem nebo telefonním číslem PSTN. [11]



Obrázek 9: *Ukázka dotazů v DUNDi 7.1 [11]*

Na obrázku 9 je ukázka dotazů v technologii DUNDi. Například budeme připojeni v síti DUNDi-test a žádáme PBX B, jestli nezná cestu na extensions 101. PBX B nám odpoví, že neví, ale zeptá se svého peera PBX D. PBX D zná cestu k danému číslu a posílá PBX B informaci o cestě. PBX B nám poté předá tuto informaci, kterou si také uloží do databáze. Nyní již můžeme poslat žádost k extensions 101 přes PBX D. DUNDi provádí vyhledávání dynamicky, a to pomocí přepínače switch v extensions.conf nebo s použitím DUNDILOOKUP(), jako dialplán funkce. PBX jsou v DUNDi identifikovány pomocí MAC adres. Důležitou hodnotou je nastavení TTL, což je vlastně počet kroků, které může DUNDi vykonat z určité PBX. Při nastavení TTL=5 se muže dotázat sousední PBX na další 4 PBX. Všechny důležité konfigurace se provádí v souboru dundi.conf. Systém DUNDi je spíše vhodný pro menší sítě, kde se rychleji spojí s ostatními uživateli. Jeho výhoda spočívá, že se dokáže spojit s pomocí ostatních PBX k uživatelům, které doposud neznal. Nevýhodou naopak může být použití ve velké síti, kde se bude muset dotazovat přes mnoho uživatelů, což bude znamenat větší časovou náročnost. [11]

Moje navrhované řešení je bližší systému DUNDi. V mém řešení se také dotazují jednotlivé PBX mezi sebou a ukládají si dané informace do databáze Asterisku. Funguje zde také dotazování přes PBX, zdali nezná VoIP cestu s hledanou pobočkou. Velká výhoda oproti DUNDi je že po spojení všech PBX, obsahuje každá PBX kompletní databázi všech Asterisků. Další výhoda mého řešení spočívá, že po spuštění skriptu jak na straně volajícího tak volaného si jejich Asterisky vymění důležité informace a následně mohou spolu komunikovat. Uživatel nemusí řešit, žádné zapisování a předávání informacích. Nevýhoda může spočívat v nedokonalé zabezpečenosti. Neznámí účastnící by se mohli dostat do této sítě a zneužít některá data.

8 Závěr

Cílem této bakalářské práce byl návrh Autokonfiguračního směrovacího systému pro SW PBX Asterisk. Při návrhu tohoto systému jsem řešil jednotlivé problémy a navrhnul pro ně výchozí řešení. Všechny tyto problémy jsem detailně rozepsal a zdárně vyřešil. Pro tento návrh jsem upravil konfigurační soubor sip.conf. Tento soubor je zde podrobně rozepsán. Po samostatném návrhu jsem vytvořil simulaci tohoto systému, abych ověřil, že mnou navržený systém bude opravdu fungovat. Simulace byla provedena na vhodné experimentální topologii. Po prvotních nezdarech se simulace zdařila a vyzkoušel jsem také funkčnost mého autokonfiguračního systému. V práci je obsažena i ukázka dialplánu pro navrhované řešení s následným vysvětlením. Po úspěšném návrhu a simulaci jsem porovnal mnou navrhované řešení s technologiemi ENUM a DUNDi.

Přínos této bakalářské práce zcela jasně vyznívá v návrhu autokonfiguračního směrovacího systému, který je zde podrobně rozepsán. Velkým přínosem je simulace tohoto systému, která je obsažena v této práci. V příloze jsou umístěny diagramy pro návrh tohoto systému i jednotlivé skripty z mé simulace. Výhodou tohoto systému je, že po spojení se všemi PBX obsahuje každá PBX kompletní databázi všech Asterisků. Uživatel se tedy může dovolat na kteroukoliv pobočku na dané PBX. Díky tomuto systému se nemusí uživatel starat o žádnou výměnu informací. Stačí mít pouze spuštěn tento systém, který veškerou potřebnou výměnu informací řeší za nás. Nevýhodou tohoto systému může být jeho bezpečnost. Jelikož šlo o návrh tohoto systému, zvolil jsem všude stejné jednoduché heslo. Tato bakalářská práce může posloužit jako dobrý základ pro zdokonalení tohoto systému.

Použitá literatura

[1] Úvod do VoIP. *MB DATA* [online]. [cit. 2013-02-20]. Dostupné z: http://www.mbdata.cz/uvoddovoip.htm#_Využití_IP_telefonie

[2] VOZŇÁK, Miroslav. *Voice over IP*. 1. vyd. Ostrava: VŠB - Technická univerzita Ostrava, 2008, 176 s. ISBN 978-80-248-1828-3.

[3] Co je to VoIP?: Telefonování 0\02 internet. *PTS telekomunikace* [online]. © 2013 [cit. 2013-02-20]. Dostupné z: http://www.ptslbc.cz/operatori/voip-pts/co-je-to-voip

[4] VoIP. *JOYCE* [online]. © 2013 [cit. 2013-02-20]. Dostupné z: http://www.joyce.cz/cz/rozcestnik/voip/

[5] VoIP, IAX protokol, 2.díl. *OWEBU: o internetu, počítačích a webhostingu* [online]. 2013 © [cit. 2013-02-20]. Dostupné z: http://owebu.bloger.cz/PC-site/VoIP-IAX-protokol-2-dil

[6] VoIP, H.323 protokol a ostatní, 4.díl. *OWEBU: o internetu, počítačích a webhostingu* [online]. 2013 © [cit. 2013-02-20]. Dostupné z: http://owebu.bloger.cz/PC-site/VoIP-H-323-protokol-a-ostatni-4-dil

[7] VoIP, SIP protokol, 3.díl. *OWEBU: o internetu, počítačích a webhostingu* [online]. 2013 © [cit. 2013-02-20]. Dostupné z: http://owebu.bloger.cz/PC-site/VoIP-SIP-protokol-3-dil

[8] TELEFONNÍ ÚSTŘEDNY ASTERISK. VOZŇÁK, Miroslav. *Teorie a praxe IP telefonie* [online]. © 2012 [cit. 2013-02-20]. Dostupné z: http://www.ip-telefon.cz/archiv/dok_osta/ipt-2008_Telefonni_ustredny_Asterisk.pdf

[9] Asterisk Open source PBX. WIJA, Tomáš, David ZUKAL a Miroslav VOZŇÁK. *Miroslav Vozňák* [online]. 15.12.2005 [cit. 2013-02-20]. Dostupné z: http://homel.vsb.cz/~voz29/files/voz_72.pdf

[10] Hardware. *Asterisk* [online]. © 2013 [cit. 2013-02-20]. Dostupné z: http://www.asterisk.org/products/hardware

[11] LEIF MADSEN, Jim Van Meggelen. *Asterisk: the definitive guide* [online]. 3rd ed. Sebastopol, CA: O'Reilly Media, Inc, © 2011 [cit. 2013-02-20]. ISBN 978-059-6517-342. Dostupné z: http://www.asteriskdocs.org/en/3rd_Edition/asterisk-book-html/asterisk-book.html#Architecture_id294803

[12] Konfigurace Asterisku (3) - extensions.conf. *Http://www.telegro.cz/* [online]. © 2009 [cit. 2013-02-20]. Dostupné z: http://www.telegro.cz/2009/04/12/konfigurace-asterisku-3-extensionsconf

[13] VoIP, kodeky - přehled, 6.díl. *OWEBU: o internetu, počítačích a webhostingu* [online]. © 2013 [cit. 2013-03-09]. Dostupné z: http://owebu.bloger.cz/PC-site/VoIP-kodeky-prehled-6-dil

[14] ENUM – konec minutových poplatků. TŮMA, Pavel. *O počítačích, IT a internetu - Živě.cz* [online]. 17.10.2007 [cit. 2013-04-27]. Dostupné z: http://www.zive.cz/Clanky/ENUM--konec-minutovych-poplatku/sc-3-a-138560/default.aspx

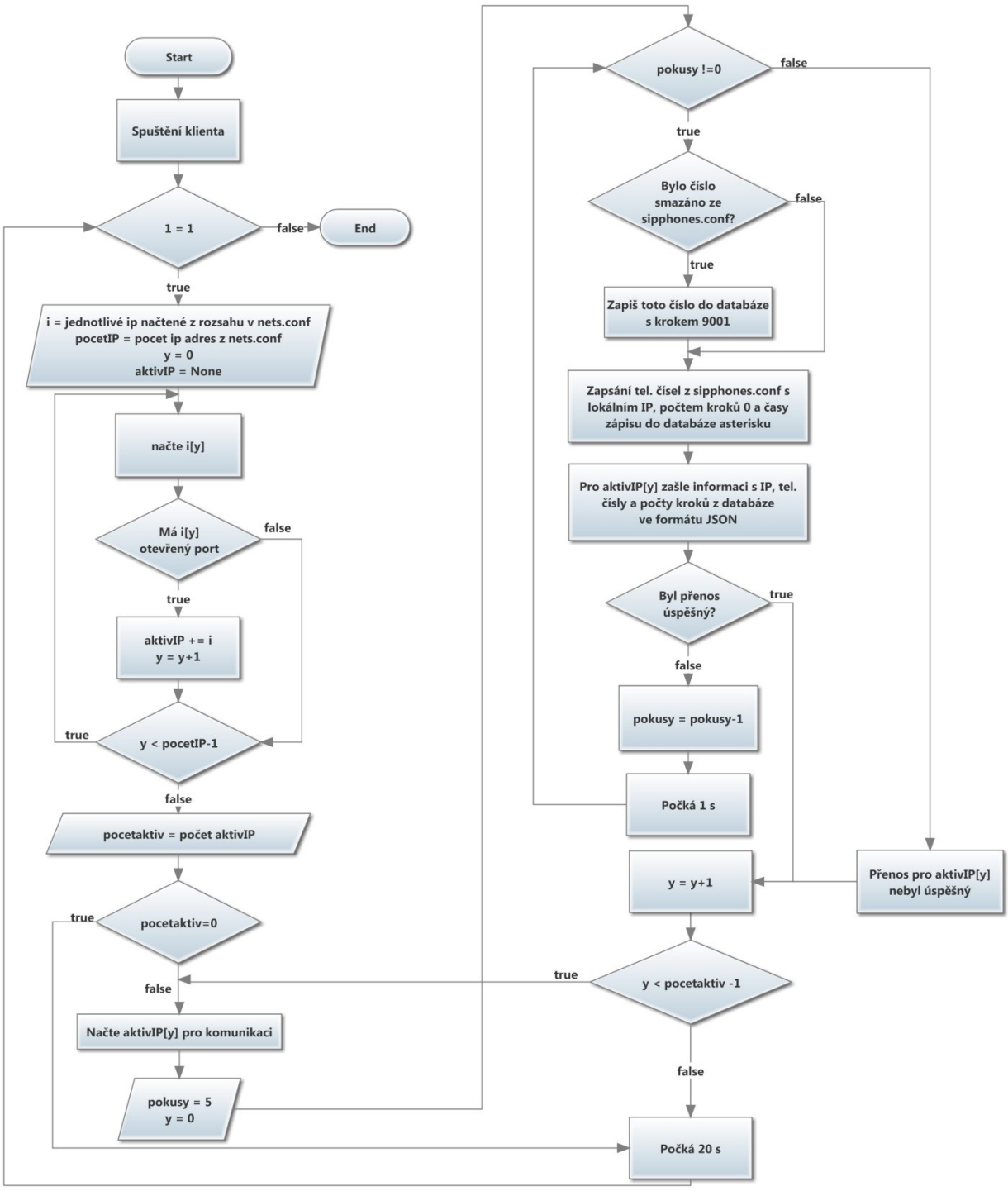
Seznam příloh

Součástí BP je CD.

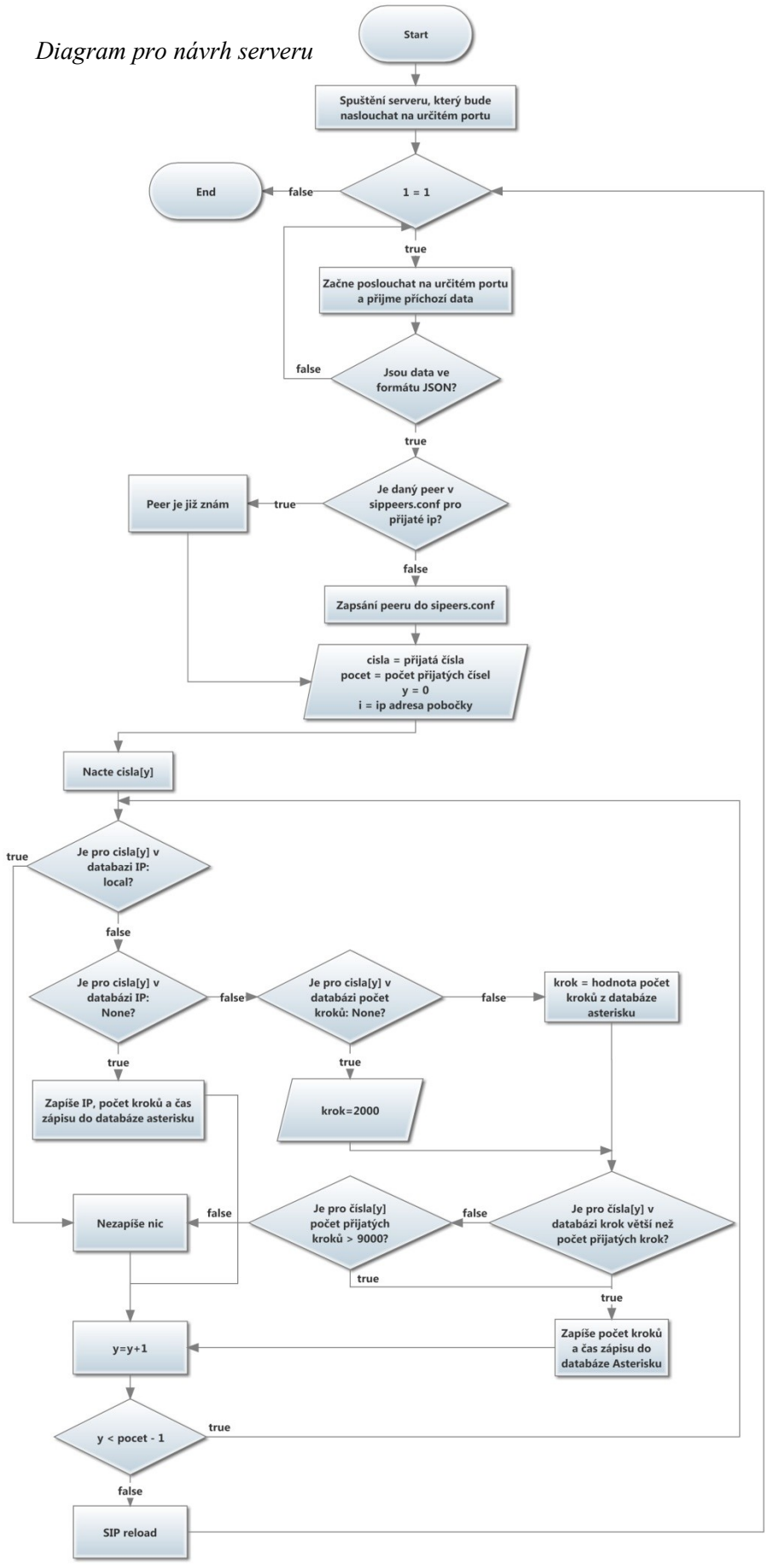
Adresářová struktura přiloženého CD:

- Diagramy/klient.jpeg
- Diagramy/server.jpeg
- Diagramy/hlídač.jpeg
- Skripty/client.sh
- Skripty/sendip.py
- Skripty/server.sh
- Skripty/clientdata.py
- Skripty/watch.py
- Skripty/parse.py
- Skripty/nets.conf
- BP/RYK0005.pdf

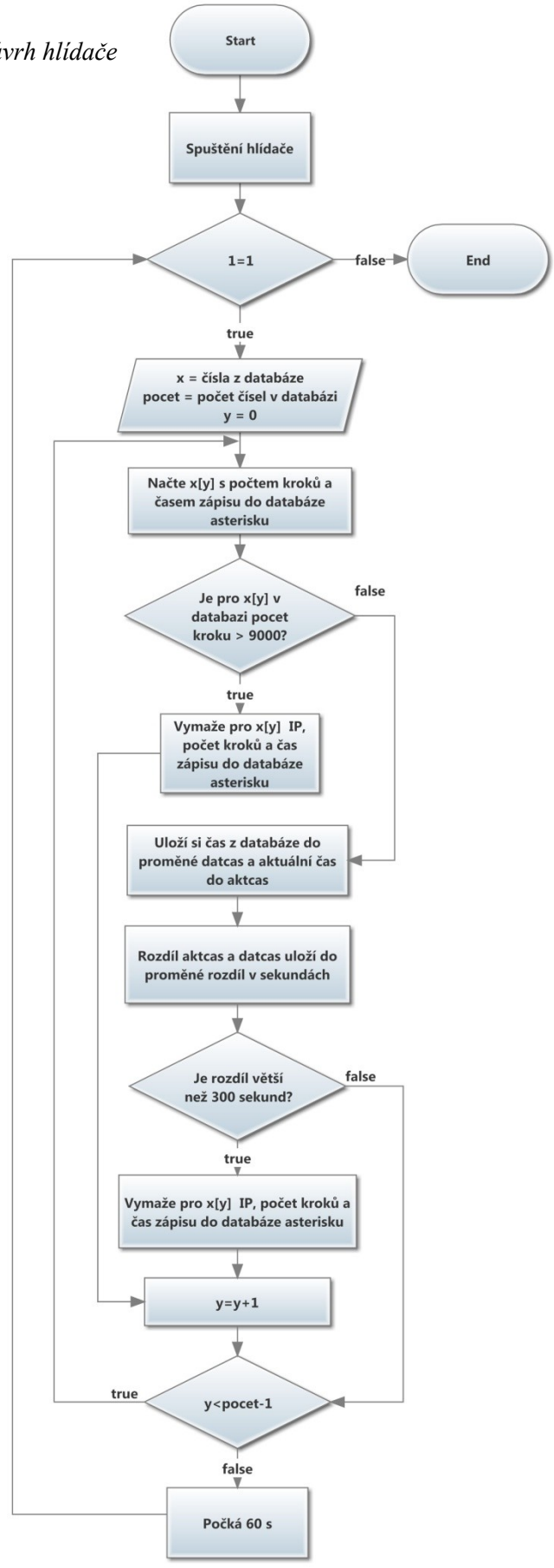
Příloha.A: *Diagram pro návrh klienta*



Příloha.B: *Diagram pro návrh serveru*



Příloha.C: *Diagram pro návrh hlídače*



Příloha.D: *Hlavní skript klienta: client.sh*

```
#!/bin/bash


while [ 1 ]; do
    active_ip=$(nmap -p 10000 $(python /home/student/aplikace/parse.py)
| grep -B 3 "open" | grep -o '[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-
9]\{1,3\}')
    # projde vsechny ip adresy z nets.conf a hleda ip adresy s otevrenym portem
10000, ktere ulozi do promenne active_ip
    for i in $active_ip; do  # pro kazdou z aktivnich ip budu posílat informace
        process_status="1"   # promenna na overeni zda prenos probehl uspesne,
pokud je 0 prenos byl uspesny
        tries="5"            # promenna pro pokusy, maximalne 5 pokusu
        while [ "$process_status" != "0" -a "$tries" != "0" ]; do
            echo $(python /home/student/aplikace/sendip.py) | nc $i 10000
 # vlozeni skriptu v pythonu, ktery zpracovava posilane informace
            process_status=$?              # zjisteni zda byl prenos uspesny
            tries=$((tries-1)              # snizeni pokusu o hodnotu 1
            sleep 1                        # prodleva 1 sekundu
        done
    done
    sleep 20
done
```

Příloha.E: *Skript sendip.py pro vytváření posílaných informací*

```
!/usr/bin/env python

import ConfigParser
import json
import subprocess as sp
from datetime import datetime
import re

def getInfo(config, section=None, option=None):
# funkce pro ziskani informace z konfiguracniho souboru
    import ConfigParser
    Config = None
    Config = ConfigParser.ConfigParser()
    if section and option:
        Config.read(config)
        optval = Config.get(section, option)
        return optval
    elif not section and not option:
        Config.read(config)
        seclist = Config.sections()
        return seclist

def dbwrite(key,number,value): # funkce pro zapis do databaze Asterisku
    for num in number: # prochazeni cisel
        try:
            p = sp.Popen(["asterisk", "-rx", "database put routing %s/%s %s" %
(key,num,value)], stdout=sp.PIPE)
            p.communicate() # Popen kvuli neocekavanemu vystupu Updated
database.....
        except:
            print "stala se chyba pri zapisu"
    return num
```

```
def dbread(key, number=None): # funkce pro cteni z databaze Asterisku
    if number: # pokud je zadan i parametr number
        try:
            retval = sp.check_output(["asterisk", "-rx", "database get routing
%s/%s" % (key,number)])
        except:
            print "stala se chyba pri cteni"
           retval = None
    else:      # pri zadani pouze parametru key
        outnumbers = {}
        try:
            retval = sp.check_output(["asterisk", "-rx", "database show routing %s"
% (key)])
            retval = re.findall("\d+\s+\:\s+\d+", retval)
 # vybrani pomoci regularnich vyrazu
            for data in retval:
                data = data.split(":")              # rozdělění pomocí znaku :
                outnumbers[data[0].strip()]=data[1].strip()
# odstraneni netisknutelnych znaku pomoci strip
            retval = outnumbers
        except:
            print "stala se chyba pri cteni"
            retval = None
    return retval
########################################################################
if __name__ == '__main__':
    my_ip = getInfo("/etc/asterisk/sip.conf", section="general",
option="udpbindaddr")  # ziskani ip adresy ze sip.conf
    nums = getInfo("/etc/asterisk/sipphones.conf")
# ziskani cisel peeru ze sipphones.conf
    d = datetime.now().strftime("%Y%m%d%H%M%S")
# ulozi do promenne aktualni cas
    dbwrite('host', nums,'local')
# zapsani informaci do databaze
    dbwrite('step', nums,'0')
    dbwrite('ts', nums,d)
    outdict = {"ip":my_ip, "numbers": dbread('step')}
# ulozeni informaci do slovniku
    jason=json.dumps(outdict)
# prevedeni slovniku do formatu JSON
```

Příloha.F: *Hlavní skript serveru: server.sh*

```
#!/bin.bash

while [ 1 ]; do
    content=$(nc -lp 10000)    # posloucha na urcitem portu
    echo $content | python /home/student/aplikace/clientdata.py &
  # pomoci skriptu v pythonu zpracuje prijate informace
done
```

Příloha.G: *Skript clientada.py pro zpracování přijatých informací na serveru*

```
#!/usr/bin/env python

import sys
import json
import subprocess as sp
import time
from datetime import datetime
import re

def getInfo(config, section=None, option=None):
 # funkce pro ziskani informace z konfiguracniho souboru
    import ConfigParser
    Config = None
    Config = ConfigParser.ConfigParser()
    if section and option:
        Config.read(config)
        optval = Config.get(section, option)
        return optval
    elif not section and not option:
        Config.read(config)
        seclist = Config.sections()
        return seclist
def sipreload(): # funkce pro reloadnuti nastaveni SIP
    try:
        p = sp.Popen(["asterisk", "-rx", "sip reload"], stdout=sp.PIPE)
        p.communicate() # Popen kvuli neocekavanemu vystupu Updated database.....
    except:
        print "stala se chyba pri reloadnuti asterisku"
    return
def dbwrite(key,number,value): # funkce pro zapis do databaze Asterisku
    for num in number: # prochazeni cisel
        try:
            p = sp.Popen(["asterisk", "-rx", "database put routing %s/%s %s" %
(key,num,value)], stdout=sp.PIPE)
            p.communicate()
 # Popen kvuli neocekavanemu vystupu Updated database.....
```

```
        except:
            print "stala se chyba pri zapisu"
    return num

def dbread(key, number=None): # funkce pro cteni z databaze Asterisku
    if number: # pokud je zadan i parametr number
        try:
            retval = sp.check_output(["asterisk", "-rx", "database get routing
%s/%s" % (key,number)])
        except:
            print "stala se chyba pri cteni"
            retval = None
    else:      # pri zadani pouze parametru key
        outnumbers = {}
        try:
            retval = sp.check_output(["asterisk", "-rx", "database show routing %s"
% (key)])
            retval = re.findall("\d+\s+\:\s+\d+", retval)
  # vybrani pomoci regularnich vyrazu
            for data in retval:
                data = data.split(":")   # rozdělění pomocí znaku :
                outnumbers[data[0].strip()]=data[1].strip()
 # odstraneni netisknutelnych znaku pomoci strip
            retval = outnumbers
        except:
            print "stala se chyba pri cteni"
            retval = None

    return retval

def writeTemp(remote_ip,local_ip): # funkce pro zapsani peeru podle daneho vzoru
   with open('/etc/asterisk/sippeers.conf', 'a') as astfile:
        astfile.write("""
```

```
[%s]
type=peer
defaultuser=%s
secret=test
context=dbtest
disallow=all
allow=alaw
qualify=5000
host=%s
    """ % (remote_ip.replace(".","_"), local_ip.replace(".","_"), remote_ip))
################################################################################
if __name__ == '__main__':
    input_data = sys.stdin.read().strip()
    input_dict = None
    try:
        input_dict = json.loads(input_data) # prevedeni prijatych dat do slovniku
    except:
        print "Chyba pri jsonu ", input_data
    if input_dict:
        my_ip = getInfo("/etc/asterisk/sip.conf", section="general",
option="udpbindaddr") # ziskani ip adresy ze sip.conf
        peers = getInfo("/etc/asterisk/sippeers.conf")
    # ziskani cisel peeru ze sipphones.conf
        d = datetime.now().strftime("%Y%m%d%H%M%S")
 if input_dict['ip'].replace(".","_") not in peers:    # neni dany peer znam?
            writeTemp(input_dict['ip'],my_ip)          # zapsani peeru
        for num in input_dict['numbers'].keys():
            if dbread('host',num) == 'local': pass
 # je pro cislo ip local? nezapisuj nic
            elif dbread('host', num) == None:
   # je pro cislo ip prazdna hodnota?
                dbwrite('host', num,input_dict['ip'].replace(".","_"))
  # zapise informace do databaze
                dbwrite('step', num,str(int(input_dict['numbers'][num])+1))
                dbwrite('ts', num,d)
```

```
            else:
                if dbread('step',num) == None:
    # je pro cislo krok prazdna hodnota?
                    krok=2000    # do promenne ulozi vysoky pocet kroku 2000
                else:
                    krok=int(dbread('step',num))
    # do promenne ulozi pocet kroku z databaze
                if int(input_dict['numbers'][num]) < krok:
   # je pro prijate cislo pocet kroku mensi nez v promenne curentstep
                    dbwrite('step', num,str(int(input_dict['numbers'][num])+1))
# zapise informace do databaze a prida prectene hodnote +1 krok
                    dbwrite('ts', num,d)
        sipreload()         # reloadnuti nastaveni SIP
```

Příloha.H: *Hlavní skript hlídače: watch.py*

```
#!/usr/bin/env python

import subprocess as sp
import re
from datetime import datetime
import time

def dbread(key, number=None): # funkce pro cteni z databaze Asterisku
    if number: # pokud je zadan i parametr number
        try:
            retval = sp.check_output(["asterisk", "-rx", "database get routing
%s/%s" % (key,number)])
        except:
            print "stala se chyba pri cteni"
            retval = None
    else:      # pri zadani pouze parametru key
        outnumbers = {}
        try:
            retval = sp.check_output(["asterisk", "-rx", "database show routing %s"
% (key)])
            retval = re.findall("\d+\s+\:\s+\d+", retval)
  # vybrani pomoci regularnich vyrazu
            for data in retval:
                data = data.split(":")     # rozdělění pomocí znaku :
                outnumbers[data[0].strip()]=data[1].strip()
  # odstraneni netisknutelnych znaku pomoci strip
            retval = outnumbers
        except:
            print "stala se chyba pri cteni"
            retval = None
    return retval
```

```
def dbdel(key, number): # funkce pro mazani informaci z databaze
    try:
        sp.call(["asterisk", "-rx", "database del routing %s/%s" % (key,number)])
    except:
        print "stala se chyba pri mazani"
    return
#########################################################################
if __name__ == '__main__':
    while 1:
        tsdict = dbread('ts')    # ulozi do slovniku cisla s casy z databaze

        for num in tsdict:       # prochazeni jednotlivych cisel
            datdate = datetime.strptime(tsdict[num],"%Y%m%d%H%M%S")
   # nacteni casu do promenne pro prave prochazene cislo
            d = datetime.now      # ulozeni aktualniho cisla
            delta = d-datdate     # rozdil mezi casem z databaze a aktualnim casem
            a = delta.seconds     # prevod casu na sekundy
            if a  >  300:
                dbdel('ts', num)  # vymazani informaci pro dane cislo
                dbdel('host', num)
                dbdel('step', num)
        time.sleep(60)            # 60 sekundova pauza
```

Příloha.I: *Skript parse.py pro výběr IP adres*

```
#!/usr/bin/env python

import re
import ConfigParser as CP

config = CP.ConfigParser()
config.read('/home/student/aplikace/nets.conf')

for i in config.get('nets', 'net').strip().split(","):
# prochazeni ip adres, stripnuti prazdnych znaku a rozdeleni pomoci carky
    print i
```

Příloha.J: *Konfigurační soubor nets.conf, kde jsou uloženy IP adresy*

```
[nets]
net=158.196.244.169/32,158.196.244.170/32
```